MASTER'S THESIS | LUND UNIVERSITY 2014

Bluetooth Low Energy in an embedded system

Johan Anderholm

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2014-18

Bluetooth Low Energy in an embedded system

(Using wireless sensors in Axis cameras)

Johan Anderholm
Johan.Anderholm@gmail.com

16th November 2014

Master's thesis work carried out at Axis Communications AB.

Supervisor: Jimmy Rubin, Jimmy.Rubin@axis.com

Examiner: Flavius Gruian, Flavius.Gruian@cs.lth.se

Abstract

All around us there is data that could be gathered and processed. The problem is to gather it. Bluetooth Low Energy is a standard that promises cheap battery driven wireless sensors with low energy consumption capable of operating for months on small batteries. For many use cases such a device seem like a perfect fit.

Bluetooth LE has been around for a few years and is now available in most new smartphones. However, there are few practical examples of it being used outside of the smartphone market. The purpose of this thesis is to evaluate the BLE standard and find a way of utilizing BLE sensors together with Axis cameras. This is done by implementing a prototype system capable of communicating with such a sensor and integrating it into the camera event system.

Finally, an existing BLE sensor is adapted and tuned to be capable of reliably communicating events such as *door opened* within a range of around 20 meters and acting upon these in a camera. The estimated run time of the sensor on two standard AA batteries is around 430 days. With this as basis, some generic recommendations on how to tune the parameters of BLE devices is discussed.

Keywords: MSc, BLE, Bluetooth LE, Bluetooth Low Energy, Bluetooth Smart

Acknowledgements

I would like to thank my supervisor Jimmy Rubin at Axis Communications for his support and guidance, as well as everyone else at Axis who have helped me.

I would further like to thank my examiner Flavius Gruian for the valuable feedback during the thesis.

All other thesis students at Axis are also well worth mentioning. For our discussions and for making my days shorter and more joyful, thank you.

Contents

1 Introduction **7**
1.1 Purpose and goal . . . 7
1.2 Setup . . . 8
1.2.1 Use case . . . 9
1.3 Related Work . . . 10
1.4 Contributions . . . 11

2 Background **13**
2.1 Other Wireless Technologies . . . 13
2.1.1 Bluetooth . . . 13
2.1.2 IEEE 802.15.4 ZigBee . . . 14
2.2 Bluetooth Low Energy . . . 14
2.2.1 Client/Server architecture . . . 15
2.2.2 Protocol Stack . . . 16
2.2.3 Attributes, services and profiles . . . 18
2.2.4 Single and Dual Mode . . . 18
2.2.5 Connections and parameters . . . 19
2.2.6 Security . . . 19

3 Supporting Analysis **21**
3.1 Stacks and Frameworks . . . 21
3.2 BlueZ Framework . . . 21
3.2.1 D-Bus . . . 22
3.3 CC2541 SensorTag Development Kit . . . 23
3.4 Summary . . . 24

4 Implementation **25**
4.1 BlueZ . . . 25
4.2 Client . . . 28

5 Power consumption and connection quality **31**
5.1 Latency in the System 31
5.2 Measuring power consumption 32
5.2.1 Modifications 33
5.2.2 Measurements 33
5.2.3 Calculations 34
5.2.4 Transmit Power effect on advertisement 35
5.3 Range and Packet loss 36
5.4 Using the SensorTag as a door opener detector 37

6 Discussion **39**
6.1 Latency 39
6.2 Power consumption 40
6.3 Range 41
6.4 Connection schemes for minimizing power consumption 41
6.5 Bluetooth Low Energy for wireless sensors 42
6.5.1 ZigBee 42
6.5.2 Bluetooth BR/EDR comparison 43
6.6 Better Security 43

7 Conclusions **45**
7.1 The future 46

Glossary **47**

Bibliography **49**

Appendix A Oscilloscope captures **55**

Appendix B Misc **59**

Chapter 1
Introduction

It is becoming increasingly important to leverage software and hardware solutions with external sensor data and real world events and there are many use cases where it could be useful to act on such events. Sensors such as thermostats, smoke detectors and humidity sensors are commonly used throughout homes and other areas. However, they are usually highly specialized and are used in a one sensor per use basis. Such sensor data may well prove to be of use for other end users as well. Physically connecting sensors using cables requires work and planning, more so than wirelessly connecting them.

This thesis will investigate how the low energy part of the Bluetooth 4.0 standard specifically known as *Bluetooth Low Energy*, or *Bluetooth Smart*, can be used to wirelessly communicate external sensor data and events to a system. Bluetooth LE is commonly advertised as a way to run wireless systems for weeks, months or even years on a battery power and according to Bluetooth Special Interest Group, two billion Bluetooth Low Energy enabled smart phone will be shipped by 2018. With this in mind it seems likely that Bluetooth LE ith its low energy capabilities will find a way into many devices where a low ower consumption is a requirement [35].

Axis Communications is primarily focused on developing network connected surveillance cameras. Modern surveillance cameras might be an application that could benefit from external sensor data and if Bluetooth LE succeeds there will likely be a wide range of wireless sensors available that potentially could be used by cameras. Axis cameras use a firmware based on Linux which will be the platform used in this thesis.

1.1 Purpose and goal

Wireless devices are sometimes preferred over devices requiring a permanent cable installation because of the flexibility it gives. On many devices such as mobile phones or surveillance cameras there is also often a shortage of physical IO ports which further give wireless connections an advantage. To fully take advantage of the flexibility given by wireless devices, that is to make them completely wireless, they need to be able to operate on battery power and thus use sufficiently little energy.

A possible disadvantage of using wireless devices compared to wired ones is that it can be harder to protect against tampering by external intruders. Wired devices can be protected by making sure physical access to the wires are restricted. Function of wireless devices on the other hand can be disrupted by jamming the frequencies used from a distance. Many types of sensor devices, such as humidity or temperature sensors, has little data to conceal but there exist scenarios where it might be important, especially so in the security industry.

Battery driven sensors could be used together with a surveillance system in places such as doors. If a sensor detects a door opening a pan, tilt and zoom camera might point itself towards the door and start recording. This is only one scenario where the availability of external data could be beneficial. However, if any such case or similar one is to work in the long term without having to replace the battery too often, it has to consume little power.

The overall goal of this thesis is to investigate if Bluetooth Low Energy (BLE) can be used to achieve what is mentioned above, that is to obtain reliable data for prolonged time from resource constrained sensor devices, and how such a solution can be realized in an Axis camera.

To further answer this it is necessary to answer a few sub questions.

- Is the Bluetooth Low Energy solution satisfactory in terms of latency?
- Is the performance of low energy peripherals, in terms of energy consumption and range, acceptable?
- Does Bluetooth Low Energy provide sufficient security with regards to protecting the data wirelessly transmitted?

In order to answer the questions, a software solution allowing a Bluetooth LE enabled sensor device to communicate with a camera is developed and integrated into the event system of a camera. Power consumption and latency is measured, and the security features of BLE are studied.

1.2 Setup

In order to communicate with a Low Energy device, a radio is required. Commonly these are available in the format of a Bluetooth Dongle, capable of both BR/EDR and BLE communication. Axis cameras typically does not contain standard USB ports and implementing hardware support for a radio is a large task. For this reason not everything will be implemented on a camera. Instead, we use a Raspberry Pi as a proxy to which the USB Bluetooth Dongle is connected. The Axis camera communicates with the Bluetooth Radio over a standard ethernet connection.

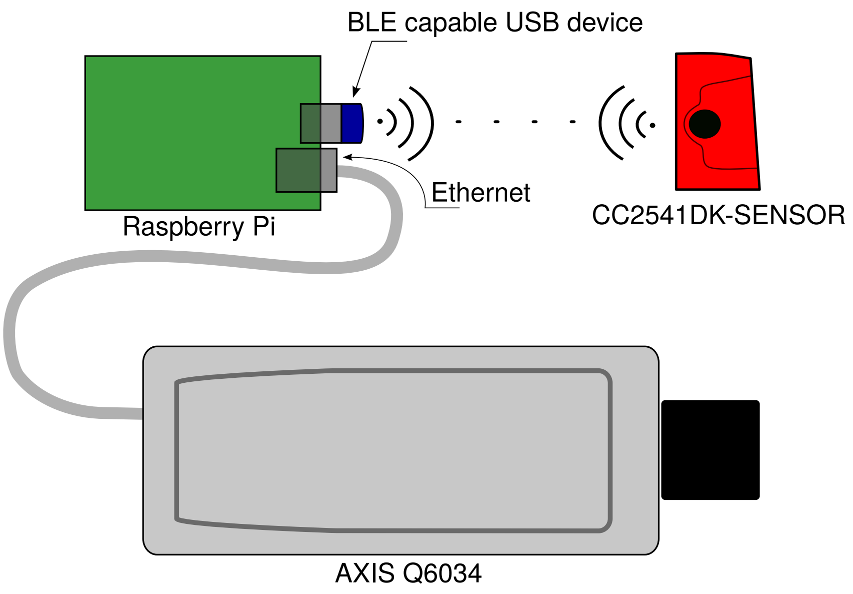

Figure 1.1: Final prototype setup..

The final prototype setup, illustrated in Figure 1.1, consists of:

- Texas Instruments CC2541 SensorTag Development Kit (CC2541DK-SENSOR)
- Raspberry Pi [31]
- Broadcom BCM20702 based Bluetooth dongle (Trust Bluetooth 4.0 USB Adapter)
- AXIS Q6034 [3]

The camera in question is a so called pan, tilt and zoom camera which is capable of pointing the camera head in all directions.

1.2.1 Use case

As a result of this thesis the AXIS Q6034 will be able to communicate with the SensorTag and use the information gathered for its internal event system. It could then act on this and do things like start recording or point the camera towards a predefined target on a push of a button or when a sufficiently large movement is noted in the gyroscope of the SensorTag.

Bluetooth Low Energy could be used for many kinds of different sensors, but in order to evaluate what BLE in sensors are capable of and how support can be implemented, a prototype door open detector will be constructed. The BLE Sensor, with a gyroscope previously mentioned, will be installed on a door and when the system registers a movement in the gyroscope sensor an event will be dispatched inside the camera.

1.3 Related Work

There exist some papers, such as [29], that study the use of BLE in wireless sensors, and further papers evaluating the use of BLE in implantable sensor systems [5]. However, these only focus on the server side, that is the peripherals aimed to be as low powered as possible. Little exist on how communication between a general master, such as computer or other relatively high power device, and a BLE peripheral. Often a specialized master device is implemented or a general Windows development tool is used to issue commands to a server device.

For mobile devices there exist well documented APIs and libraries that enable writing applications for BLE devices[21, 20]. But these APIs and libraries are closely tied to their corresponding operating systems. There exist many Linux based devices that does not use Android that cannot use those APIs.

For a competing technology called ZigBee there exist a paper on consumption modeling [10] that is used for comparing power consumption.

One of the most important, if not the most important, aspects of battery driven wireless sensors is their power consumption. In a thesis conducted at Lund University [26], the authors evaluated the performance of a wireless device using IEEE 802.15.4 with 6LoWPAN and CoAP built on top of the Contiki operating system. In short these techniques is similar to how HTTP servers work but for resource constrained devices. That thesis resulted in a functional wireless sensor but with a lifetime of no more than a few weeks on AA batteries. While their solution arguably is very flexible using IP and CoAP, the battery lifetime is not acceptable for wireless sensors in a camera surveillance setting. 6LoWPAN is a technique for allowing IPv6 packets to be sent to IEEE 802.15.4 devices and is adapted for small size but there is still a possibly an unnecessary overhead for many applications. The BLE standard has been optimized to reduce addressing overhead as much as possible and therefor should allow a lower energy consumption.

In the paper by Mackensen, Lai and Wendt [29] BLE is used to control the toy cars in a slot car racing track. The cars was equipped with a gyroscope, accelerometer and a BLE radio which reported readings to a nearby computer where it was further analyzed. Bluetooth Low Energy turned out to be a good fit for that particular problem. The findings of that study cannot be directly applied to out problem because:

- The fast cars on the race track required a high sampling speed, we do not need anywhere near that to detect if a regular door opens.
- The toy cars are provided virtually unlimited power from the rails in the race track. We need to operate on batteries.
- No regard to range was taken. If sensors are to be attached to a camera, a certain range is required.

Ali, Albasha and Al-Nashash [5] used BLE in an implantable glucose monitoring system. The prototype was meant to be powered inductively through the skin. Measurements where sent to a nearby receiver. Again, the range required is not comparable to the one we need. In the glucose monitoring system transmissions are only required to reach within a few meters, to a mobile phone or equivalent, from the patient.

1.4 Contributions

In order to realize the prototype described in the previous section, a few tasks will be carried out.

- Investigate how BLE support could be realized in a Linux environment as well as select a suitable way to implement it.
- Implement a software capable of communicating with a BLE sensor from a camera.
- Implement sensor services in a bluetooth framework.
- Tune a Bluetooth Low Energy device for a particular use case. The use case being to detect if a door has been opened.

At the end of this thesis we hope to have decided whether Bluetooth Low Energy is a reasonable choice when designing battery driven sensors for detecting simple events such as if a door has opened.

Chapter 2
Background

The 2.4GHz industrial, scientific and medical (ISM) band is used by several wireless standards because it is almost globally free for unlicensed use [27]. In this chapter we will provide an insight into some of the standards which operate in this band before going further into details on Bluetooth Low Energy.

2.1 Other Wireless Technologies

There are several standards available. Some such as IEEE 802.11 and all of its variants are focused at creating wireless local area networks with low latency and high throughput while other are focused at minimizing the power consumption. We will not cover IEEE 802.11 but instead focus on some standards more suitable for low power communication. There are of course many such standards available but we have restricted ourselves to the following few:

- Bluetooth
- Bluetooth Low Energy
- IEEE 802.15.4 (ZigBee)

2.1.1 Bluetooth

The Bluetooth standard, or Bluetooth BR/EDR, was originally designed as a way to replace cables. This could be RS-232 cables, audio and input cables for devices such as mice and keyboards. It is included in most mobile phones and is today one of the most common ways to communicate between mobile phones and wireless accessories. Bluetooth uses a total or 79 frequencies each 1MHz wide and utilizes adaptive frequency hopping to quickly hop between channels and in this way be more robust to disturbances in certain channels. The

Basic Rate and Enhanced Data Rate (BR/EDR) are rated at 2Mbps and 3Mbps respectively [1].

2.1.2 IEEE 802.15.4 ZigBee

IEEE 802.15.4 is a physical and media access control used by for example ZigBee. The ZigBee standard is low power and has a mesh networking architecture. Mesh networking makes the technology suitable for wide sensor networks that could span entire buildings. For mesh networking to work properly, all devices on the route to the destination is required to relay the data. In a paper comparing IEEE 802.15.4 with Bluetooth Low Energy it is stated that the energy utilization of BLE is 2.5 times better than IEEE 802.15.4 [34]. Adding the fact that mesh networked devices need to relay more data than a device directly connected to its destination makes this equation worse for a single low powered device.

ZigBee devices do not use channel hopping but instead rely on a single frequency when communicating to each other. This means some care has to be taken when setting a network up so that it does not collide with other wireless communication. It does however support switching channels in case of interference on the selected frequencies [12].

2.2 Bluetooth Low Energy

The Bluetooth 4.0 Standard [7] was released 2010 and specifies a new version of the Bluetooth technology that as IEEE 802.15.4 is targeted towards wireless devices requiring a lower power consumption. The technology was originally developed by Nokia under the name Wibree but was adopted by the Bluetooth Special Interest Group for the Bluetooth 4.0 standard. It is commonly referred to as *Bluetooth Low Energy*, *Bluetooth LE* and *Bluetooth Smart* and was designed so that it may share many of the components of Bluetooth BR/EDR, such as the antenna, but is a completely new technology and is not backward compatible with Bluetooth BR/EDR. Other things such as the L2CAP layer also originate from Bluetooth BR/EDR but has been modified to better fit the energy requirements of Bluetooth LE.

To facilitate a wide range of use cases, as well as minimizing power consumption, there are a few different types of BLE devices:

Peripheral Advertises its existence and may be connected to by a central device. Can both send and receive.

Central Commonly has less tight power consumption requirements. The Central may connect to and communicate with Peripheral devices as well as sending and receiving data.

Broadcaster Not connectable, advertises data to any scanning LE capable radio nearby. Is not required to receive data.

Observer May only listen for other devices. Is not required to receive data.

Bluetooth LE distinguishes between channels used for advertising information of a LE device and channels used for later communication. Listeners only have to listen to three

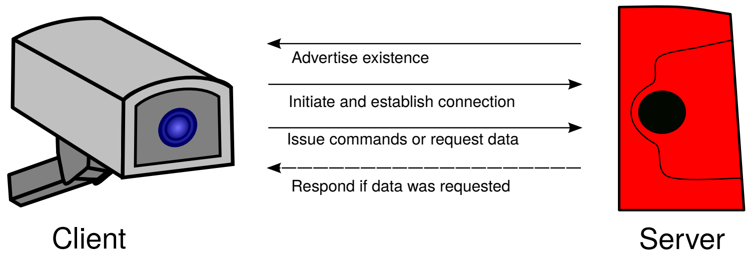
Advertise existence
Initiate and establish connection
Issue commands or request data
Respond if data was requested
Client
Server

Figure 2.1: Bluetooth Low Energy is commonly used in client/server architecture. The resource constrained device advertises its existence as well as connection details and thereby allow listening clients to connect and use it.

special advertisement channels, distributed evenly over a total of 40 channels, before deciding to connect.

After a connection is established, 37 channels are used for data transmission. These do not include the three advertisement channels. Data is transmitted in a frequency hopping manner where each data packet is sent on a new frequency. If a frequency cannot be used for transmission due to interference the packet is retransmitted at the next frequency. Channels with interference can by dynamically disabled and skipped next time. This can be compared to ZigBee which does not switch channels until interference is deemed too high and a new one channel with less interference is found [16].

A potential disadvantage of BLE compared to ZigBee is that it is connected in a star bus network topology instead of a mesh network. This means a BLE peripheral device requires a central in range where ZigBee can use other devices to reach its destination. On the other hand, if the sensors are always expected to be in range of the central device BLE could provide a lower power consumption both for the central unit and the peripheral, as stated in Subsection 2.1.2 [14].

2.2.1 Client/Server architecture

Much of Bluetooth Low Energy is built around a notion of clients and servers. A server device typically can be a resource constrained device that is connectable by clients. Clients talk to servers available by issuing data read/write requests or commands. It is also possible for the server to push data to connected clients. Servers exist to provide services, such as providing sensor measurements, to interested clients (See Figure 2.1).

Typically, a peripheral device act as the server while a central device as the client, although nothing prevents the opposite. No matter what, only peripherals, and broadcasters, sends advertisement packets and centrals or observers listen to this. A central may choose to initiate a connection to a peripheral when receiving advertisement packets from the peripheral.

2.2.2 Protocol Stack

The layers of the Bluetooth LE stack can be seen in Figure 2.2. There are some similarities to Bluetooth BR/EDR, for example both protocols use L2CAP, but they have been adapted in different ways to minimize overhead and allow for a lower power consumption.

The stack is divided in a few different components. The *Controller* part and the *Host* layers are separated by the Host Controller Interface (HCI). The HCI enables the host to control the controller in a standardized way.

This well defined split between controller, host and applications allows some flexibility in design. All parts may be implemented as a single potentially lower power chip or it may be split up in Controller/Host part that communicates with an externally implemented application. Another well defined split is to keep the Controller on one chip and Host/Application on another [17]. In the following examples the + sign indicates a combination of two components and the / sign a separation:

Controller+Host+App Very low power integrated System on a Chip (SoC) designed for peripherals such as the CC254X from Texas Instruments or nRF51822 from Nordic Semiconductors. Everything is implemented on a single chip.

Controller+Host / App Achieving potentially low power but has greater flexibility than single chip solutions.

Controller / Host+App Standard way on mobile phones, computers and other devices with powerful processors not subject to low power requirements. The Linux kernel for example uses the standardized HCI commands to communicate with a wide range of hardware but lets the rest of the operating system implement host and app layers.

Controller

At the bottom of the stack the *Physical Layer* and the *Link Layer* are found. Together these form a radio unit that can be controlled using HCI.

Bluetooth LE uses the same 2.4GHz ISM band and Gaussian Frequency Shift Keying (GFSK) modulation technique as Bluetooth BR/EDR but is different in some ways. The frequency band is divided into 40 channels with 2MHz in between each channel. Raw transmission rate is one bit per microsecond (or 1Mbps) without taking any packet overhead into account.

The Link Layer is responsible for ensuring packets sent and received packets are structured correctly and is not corrupted with regard to their checksum. The 40 channels mentioned earlier are further grouped into 3 advertising channels and 37 data channels. Advertising channels are used for broadcasting simple data, advertising the existence and connectability of a device as well as initiating connections.

A connection uses the 37 data channels in a channel-hopping fashion and enable data packets to be tracked and retransmitted as well as encrypted and authenticated. When a device wants to connect to an advertising device it does so by waiting for an advertisement and sending a *connection request* packet on the same advertisement channel [16].

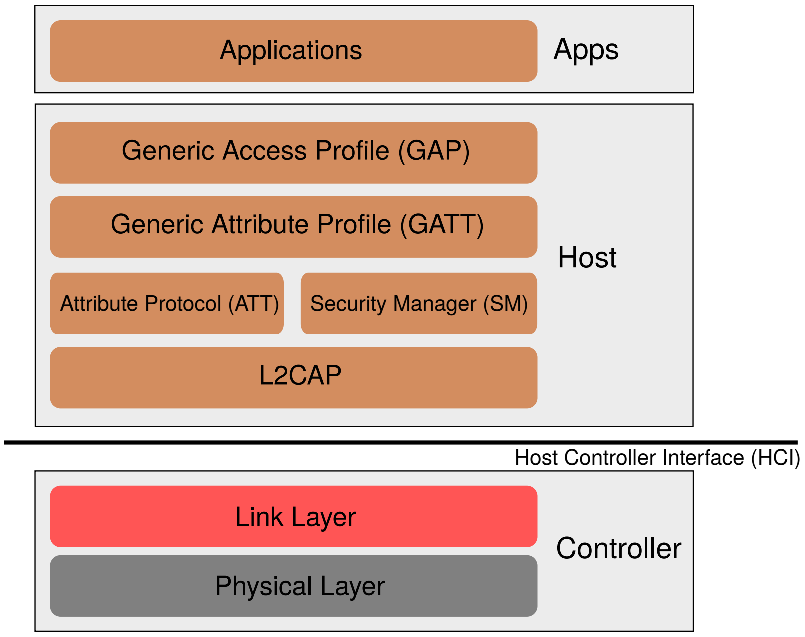

Figure 2.2: The Bluetooth LE protocol stack is split into three main parts. Controller, Host and Application layers. In actual implementations layers may be combined to decrease power consumption.

Host

On top of the controller, connected by HCI, is the host. It consists of multiplexing layers, protocols and procedures for managing higher level applications. It contains the following layers:

Logical Link Control and Adaptation Protocol (L2CAP) Performs fragmentation and defragmentation of data and defines three fixed channels for the Attribute Protocol (ATT), the LE L2CAP Signaling and the Security Manager (SM) Protocol respectively.

ATT In BLE data is communicated with so called *attributes*. Attributes are addresses in the BLE peripheral that can be written to or read from, depending on the permissions given an attribute.

SM Defines a protocol for pairing and key distribution. The security available in BLE is further described in Subsection 2.2.6.

Generic Attribute Profile (GATT) Used to further organize the attributes made available on a peripheral. Allows remote devices to discover and use the services a peripheral makes available.

Generic Access Profile (GAP) A profile to all BLE devices that define how devices can connect and be discovered.

2.2.3 Attributes, services and profiles

Peripherals in Bluetooth LE uses a concept of profiles, services and attributes to make data available to the outside world.

At the lowest level there is an attribute. It is simply an address that can be read from and written to if permitted. This is the only official way of communicating data to a BLE device. A few different types of attribute requests exist:

Requests A client device reads from or writes to an attribute and the server have to acknowledge.

Commands A central device reads from or writes to an attribute and the peripheral does not have to acknowledge.

Indications A peripheral sends an attribute value to connected central devices and requires an acknowledgment.

Notification Same as Indication but without requiring an acknowledgment.

An attribute is nothing more than an address (*handle*) tied to some value in a peripheral device. For an attribute handle to be meaningful it has to be discovered using GATT. During this process a handle is associated with an universally unique identifier (UUID), chosen during the development process, and its meaning can thereafter be known to the central device. Several attributes can be tied to one UUID.

A *Characteristic* is an attribute value. An example of this would be temperature. The temperature characteristic can be read from, although not written to. Another closely linked characteristic could be a boolean value that enabled or disables the thermometer sensor.

Related characteristics such as this can be bundled together into what is called a *Service*. The two temperature characteristics mentioned above would typically be grouped together into a temperature service.

Finally, a whole set of related services become a *Profile*.

These attributes are only accessible while being connected. Upon a connection GATT is used to discover the rest of the services. Handle addresses are not expected to change regularly and can be cached although there exists mechanisms for telling a device trying to connect that things might have changed and require a rediscovery [17].

2.2.4 Single and Dual Mode

Some Bluetooth devices implement both the BR/EDR and the LE parts of the standard while others implement only the LE specific parts. Typically, the radio chipsets used in master devices with less strict requirements on power consumption such as cellphones and computers are so called *Dual Mode* devices. They are able to communicate to both BR/EDR devices and LE devices.

Peripherals often requires very low power consumption in order to operate on battery for extended amounts of time. These have the option of operating in *Single Mode*. Single mode devices are only capable of communicating with other Single or Dual mode LE capable devices, with less power consumption as a result [17].

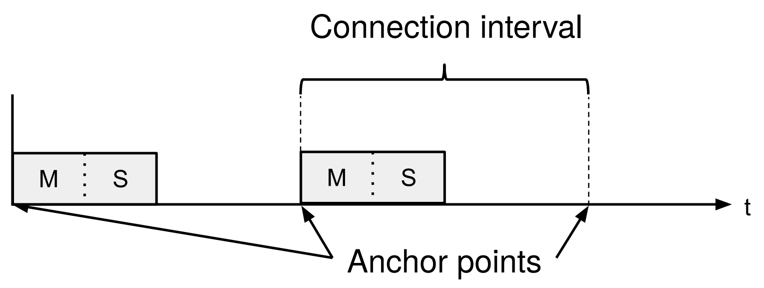

Figure 2.3: Connection events occur at anchor points. The Central master device initiates communication and the slave responds. The time between two anchor points is configured at startup. The master device must always initiate connection but the slave may skip answering if the *Connection slave latency* parameter has been agreed upon in advance.

2.2.5 Connections and parameters

In Bluetooth Low Energy data transfer occurs during specific set intervals. During a connection the peripheral and the central negotiates certain connection parameters that should be used until next negotiation.

Advertisement Interval A peripheral advertises its location and is thus available for connection at advertisement points. It advertises some data identifying it and waits for incoming data for a short amount of time.

Connection Interval In BLE data exchange occurs at specific points of time called *anchor points*. This occurs at a negotiated connection and thus both devices may be in sleep mode until an anchor point is encountered.

Connection Slave Latency The central device must always be available and broadcast a packet at each anchor point. The peripheral on the other hand might ignore the central for a negotiated interval called *Connection Slave Latency*.

These parameters can be carefully tuned as they decide both what latency there will be in the system and the power consumption. Whenever an anchor point occurs, and the slave may not skip it because of slave latency, the master makes any request it wants to or simply send an empty package which the slave responds to with data if there is any to be sent or an empty package if there is not. The protocol dictates that both devices must wake up and transmit and receive data at these negotiated anchor points. An illustration of this can be seen in Figure 2.3.

2.2.6 Security

The standard of Bluetooth Low Energy defines a Security Manager with a few different levels of security that can be further divided into a *paired* mode and a completely unauthenticated and unencrypted mode. In the paired mode devices may be authenticated and have encryption enabled, and to establish a paired mode a shared secret has to be exchanged. The shared secret is critical for further establishment of the encryption which uses AES-128. The shared secret can be exchanged by first establishing a temporary key in one of the following ways:

Just Works Use key 0.

6-pin digit Let user input 6 digits common between devices.

Out of band The two devices shares a key some other way, for example using near-field-communication.

The temporary key is used together with two random numbers provided by the master and the slave to generate a short term key which is used to encrypt packets between the two devices to let them exchange a long term key [17, 32].

A temporary key of 0 will obviously give no kind of security at all if there is anyone listening during long term key negotiation and no protection against a man-in-the-middle (MITM) attack. It is also shown in [32] that the 6-pin authentication does not protect against an eavesdropper and possibly not against a MITM attack either because a 6 digit search space is too small and is exhausted within seconds by brute force on a modern computer. One could argue that Bluetooth Low Energy is fundamentally insecure without relying on an out-of-band method. Although, in Chapter 7 a possible solution is proposed.

Chapter 3
Supporting Analysis

Bluetooth Low Energy support has been enabled by default in the Linux kernel since 2012 [18] but was available as an option earlier, from around kernel version 3.2. In order to design the system for communicating with the SensorTag and any other BLE device the available options must first be considered.

3.1 Stacks and Frameworks

There are a few different stacks available. SoCs like CC254x or nRF51822 typically come with their own stack that is well suited for that particular microcontroller.

For Linux there are two open source stacks compatible with Bluetooth Low Energy available. One called BTstack and the other BlueZ. BlueZ is known as *Official Linux Bluetooth protocol stack* [25]. BTstack is another interesting choice because it is advertised that it can run on everything from 32kB FLASH 4kB RAM without a RTOS to full desktop systems. It is also not specific for Linux but can be ran on other platforms as well [2].

3.2 BlueZ Framework

BlueZ [30] is the predominant framework used for Bluetooth communication, both BR/EDR and LE, in Linux environments. It is a collection of programs and libraries used for controlling the hardware and data links by communicating with the Linux Kernel which in turn communicates, through HCI, with the hardware adapters connected and lastly to other devices. BlueZ and the Linux kernel together can roughly be translated to the Host layer of the protocol stack in Figure 2.2.

There are two ways of using communicating and controlling bluetooth devices through BlueZ. Either through libraries or by talking to a bluetooth daemon through an interprocess communication (IPC) layer called D-Bus. The libraries can be built using configure

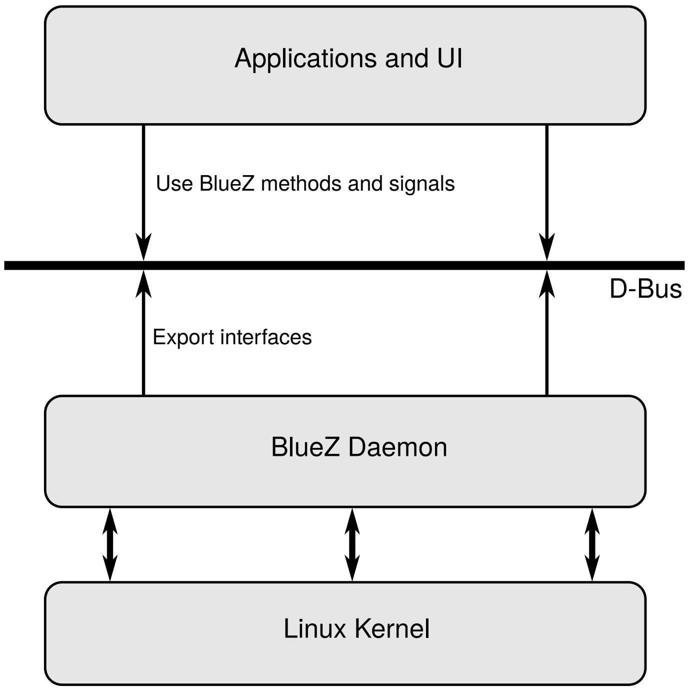

Figure 3.1: BlueZ architecture. BlueZ and the kernel drivers make up the *Host* layer and makes itself available for applications through D-Bus.

flags but this is not enabled by default [6]. Furthermore, many of the functions used to communicate with LE devices are not available through the dynamically linkable libraries. To use BlueZ together with Bluetooth LE the second approach is most feasible as it does not require rebuilding BlueZ for every program that communicates with BLE devices.

A high-level illustration of the architecture used in the second approach can be seen in Figure 3.1.

The BlueZ Daemon (called bluetoothd) is responsible for controlling connected bluetooth adapters and scanning for Bluetooth both LE and BR/EDR devices. It exports D-Bus interfaces with methods and signals that client applications can use to control adapters, view devices available for connection and connect to them. After connection the BlueZ Daemon further exports interfaces on the D-Bus for interacting with newly connected device [15].

3.2.1 D-Bus

D-Bus is an IPC system commonly used in Linux to enable separate processes to communicate with one another. At heart it is a library that allow applications to connect to a common *System Bus* or *Session Bus* and post messages and register to messages posted by other processes.

In D-Bus there are couple of concepts used:

Object Paths Applications publish *Objects* on the buses. This could be for example /org/bluez.

Methods and Signals Methods are operations that can be executed on an Object with parameters. There is always a response (although it might be empty). Signals are published by an Object and other processes can register to observe that signal. The object that registered the signal pushes a message over D-Bus when some event occurs and all processes that are observing the signal receives this message and can act upon it.

Interfaces Groups of methods and signals available on an object.

The concepts are quite similar to that of Object Oriented languages and many bindings exist to make it possible to register native objects in that particular language on D-Bus [13].

3.3 CC2541 SensorTag Development Kit

Texas Instruments have released a development kit based around their CC2541 SoC. It is a device powered by a single CR2032 battery and is already populated with several sensors of different types. It is an implementation of what is known as a *peripheral* type of device (See Section 2.2).

The device is controlled using the following list of the GATT services (See Section 2.2.3).

- GAP
- GATT
- Device Information
- IR Temperature
- Accelerometer
- Humidity
- Magnetometer
- Barometer
- Gyroscope
- Simple Keys

Apart from GAP and GATT only *Device Information* which is used to expose device information such as manufacturer, serial number, firmware version and similar data is compatible with the official SIG profiles [33]. The rest are custom manufacturer profiles. The *IR Temperature*, *Accelerometer*, *Humidity*, *Magnetometer*, *Barometer* and *Gyroscope* services are each connected to a physical sensor. *Simple Keys* is the service providing means to notify any connected devices of button presses on any of the three buttons available on the SensorTag.

The SensorTag is used by TI to showcase a typical BLE unit and to be used as reference design for smart phone accessories [9]. There exist several functional libraries and finished applications capable of communicating with the SensorTag for Android and iOS [9], but no complete Linux/BlueZ based solutions. It was chosen as peripheral in this thesis because it is a reference device and at the same time a fully functional wireless BLE device.

3.4 Summary

To summarize there are some choices for the final solution available. A few problems exist however. A radio cannot easily be connected directly to the camera without also implementing the HCI layer mentioned in Section 2.2.2 and a Linux Kernel driver to accompany it. Instead, we opted for a proxy solution where the *Bluetooth Daemon* (See Figure 3.1) executes on the proxy, that is a Raspberry Pi in this case, and communicates directly with a USB connected Bluetooth 4.0 Dual Mode radio.

The clients used to control and communicate with BlueZ, and indirectly Bluetooth LE devices, over D-Bus can be implemented on a camera. The D-Bus commands are then send to the remote part over TCP/IP instead of the regular local Unix Domain sockets. An advantage of the D-Bus approach over simply using the libraries is that it is not tied to any programming language. There exist D-Bus bindings for many languages, for example *C*, *C++*, *Java*, *Python* and many more [13]. Some potential disadvantages are that it requires D-Bus to be available on the platform which means it cannot be used in every situation. Axis however do have support for D-Bus and uses a flavor of Linux that can handle D-Bus and BlueZ. For more resource constrained devices *BTstack* or one of the vendor specific stacks could offer a better solution.

A result of our solution, with BlueZ being controlled over D-Bus and TCP/IP, is that if hardware support for Bluetooth 4.0 is implemented in camera hardware in the future no extra modifications are required. The Bluetooth Daemon simply need to be moved to the camera and the clients modified to connect to the local system D-Bus instead of the remote one.

Axis offers a platform to allow third parties to create applications for their camera systems. The Axis Camera Application Platform consists of among other things an SDK with compiler systems and libraries that can be used to interface with the event system of a camera. By using this it is possible to expose application defined events which the camera operator can use to trigger actions on. This is used in our system to propagate happenings on the SensorTag to the camera software [4].

Chapter 4

Implementation

In Chapter 3 and Chapter 2 a few different approaches and alternatives were described and summarized. This chapter further presents the solution proposed in Section 3.4.

In order to obtain sensor data from the SensorTag it is first necessary to implement the profiles mentioned in 3.3. We will implement the profiles in BlueZ and access to them will be made available through D-Bus interfaces. Furthermore, clients for controlling BlueZ and the SensorTag are necessary. A high-level illustration can be seen in Figure 4.1. The exported interfaces in the BlueZ Daemon are preexisting except for the profiles. The client part is implemented from scratch.

4.1 BlueZ

Profiles for Low Energy in BlueZ are at this time required to be statically compiled into BlueZ itself during compile time. BlueZ uses a simple plugin-based architecture where plugins are registered by using a macro (See Listing 1). In the example the `ti_gyroscope_init()` and `ti_gyroscope_exit()` functions are run when a plugin is loaded and unloaded respectively.

The registered init function further sets up a profile definition using the function `btd_profile_register(struct btd_profile *profile)`.

Listing 1: Setting up plugin *init* and *exit* functions in BlueZ.

```
BLUETOOTH_PLUGIN_DEFINE(sensortag_gyroscope, VERSION,
                BLUETOOTH_PLUGIN_PRIORITY_DEFAULT,
                ti_gyroscope_init, ti_gyroscope_exit)
```

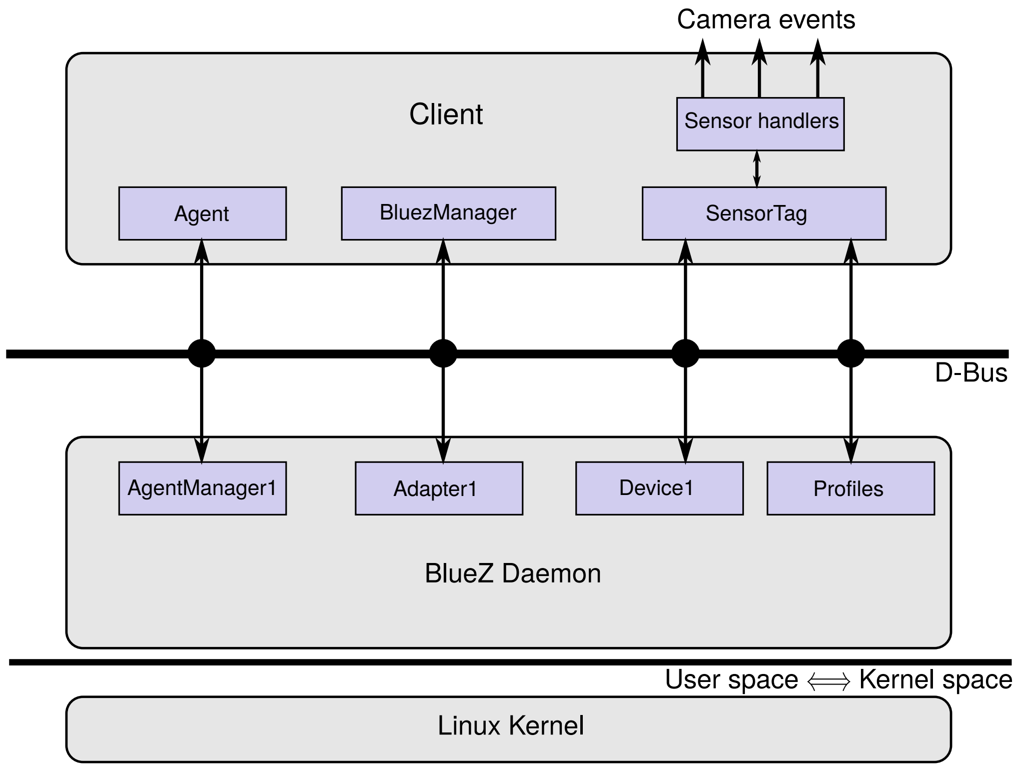

Figure 4.1: Architecture of final solution. The BlueZ Daemon is running on a Raspberry Pi and the client frontend software on the camera.

Listing 2: Plugin definition.

```
static struct btd_profile ti_gyro_profile = {
  .name = "Ti Gyroscope GATT Driver",
  .remote_uuid = "f000aa50-0451-4000-b000-000000000000",

  .device_probe = ti_gyroscope_device_probe,
  .device_remove = ti_gyroscope_device_remove,

  .adapter_probe = ti_gyroscope_adapter_probe,
  .adapter_remove = ti_gyroscope_adapter_remove,
};
```

The profile structure (See Listing 2) can be summarized as:

name Descriptive name of profile.

remote_uuid Ties this specific profile to the UUID of the service.

adapter_probe Callback executed whenever a new adapter has been connected to BlueZ. Can be used for exporting profile specific interfaces tied to a certain adapter or initializing locally exported services.

adapter_remove Used for cleaning bluez up upon removal of adapter.

device_probe Callback executed whenever a device with services corresponding to *remote_uuid* has been found. Typically, happens after service discovery after having connecting to a device.

device_remove Enables cleaning up after a device has been removed.

An adapter in this case is a radio, and a device is any kind of Bluetooth device that at some point has been discovered by BlueZ.

In the example from Section 1.2.1 where the SensorTag Gyroscope is used as a door open detector the profile implementation can be summarized as:

1. `ti_gyroscope_init` function registers the profile in Listing 2 in BlueZ.
2. Whenever a Bluetooth USB dongle (or any other radio) is detected `ti_gyroscope_adapter_probe` gets executed and sets up some internal structures.
3. A device with a service matching the gyroscope UUID is discovered and triggers `device_probe`. The function exports, on D-BUS, methods for controlling the gyroscope and signals for receiving notifications from the remote device. It also further discovers the characteristics of the gyroscope service such as enable/disable sensor and enable/disable notifications.
4. When a D-Bus method is called by a client application a local function is executed which can use the `gatt_write_char` or `gatt_read_char` functions to read or write detected attributes (See Subsection 2.2.3) on the remove device and respond to the D-Bus clients.
5. If measurement notifications are enabled the remote device pushes data to BlueZ which is handled by a previously registered handler function that processes the raw sensor data and signals the data over D-Bus.

In order to allow client applications to use the SensorTag upon connection a few D-Bus methods and signals are exported:

Enable Method that enables the gyroscope sensor on the remote device.

EnableNotification Method that enables the SensorTag to send periodic gyroscope samples.

Disable Method that disables the gyroscope sensor on the remote device.

DisableNotification Method that stop the SensorTag from sending periodic gyroscope samples.

GetAxisData Method for reading the gyroscope sensor data attribute. Raw sensor data received by BlueZ is then processed into decimal numbers.

AxisData Signal that broadcasts periodic gyroscope samples if notification has been enabled.

4.2 Client

The client that can be seen the top part of Figure 4.1 is the part that is running on an actual camera. It is an application utilizing the Axis Camera Application Platform and communicates with the following server interfaces:

AgentManager1 Used to register Agents for applications.

Adapter1 Used for controlling a radio.

Device1 Exported by BlueZ for controlling a device.

Profiles Various interfaces exported by the previously implemented profiles.

The above interfaces are exported to D-Bus and allow clients to control and use the different parts of BlueZ. Our client consists of the following parts:

BluezManager Responsible for controlling the adapters connected to BlueZ and for discovering SensorTag devices and connecting to them. May also start pairing and such things.

Agent Whenever a paring request is started the Agent is responsible for displaying any pin codes and allow for entering them.

SensorTag Responsible for managing the actual SensorTag device and does things such as connecting and making sure it reconnects in case of connection loss.

Sensor handlers Receives sensor data from the remote device in form of signals. In the case of the Gyroscope it detects whether any large movements have been detected since last sample and if it is above a threshold it uses the part of the Axis development kit to create events for the rest of the camera system.

To summarize how the solution works here is a common scenario of how a SensorTag is found and used.

1. The client starts and connects to BlueZ.
2. Any adapter is powered on and set to start scanning.
3. An advertising SensorTag comes within range and is detected by BlueZ.
4. BluezManager discovers it and initiates a connection
5. When the device becomes connected any optional paring is initiated.
6. If pairing is initiated, Agent is used to exchange pincodes.
7. When the optional pairing step is completed the gyroscope and gyroscope notifications are enabled.

8. The SensorTag pushes attribute data to the gyroscope sensor handler which uses these to calculate if any sufficiently large movements have occurred and dispatches a camera event if so has happened.
9. Whatever action the camera operator has previously assigned to the door controller, that is the gyroscope, alarm is executed.

Chapter 5

Power consumption and connection quality

When designing a Bluetooth Low Energy peripheral device supposed to run on battery power, such as the door opener detector suggested in Subsection 1.2.1, one of the main goals is to minimize the power consumption while at the same time have a sampling rate and latency allowing the particular use case.

In the door opening detector example the sample rate of the gyroscope has to be sufficiently large to detect any movements and the latency from when a sample is made until it reaches the camera has to be low enough for the camera to react in a timely manner. At the same time a high sample rate and a short period between anchor points (See Subsection 2.2.5) will affect the battery longevity negatively. In this chapter different aspects of the parameters of Bluetooth Low Energy are investigated and we will find out how a BLE device can be tuned for good performance in terms of power consumption and latency.

5.1 Latency in the System

Latency is mostly affected by the interval parameters described in Subsection 2.2.5. Data transfer has to occur at the specific anchor points dictated by the connection interval. Thus, the connection interval is the parameter that is most important for the latency of communication.

There are two main methods for attribute value transfer dictated by the ATT protocol (See Subsection 2.2.3). Requests are initiated by the client while notifications are initiated by the server. This means the latency for retrieving values differs and has to be measured differently.

To measure the time between a notification occurs at the server until it arrives and can be processed by the client a simple circuit, that can be seen in Figure B.1 in Appendix B, was used. A GPIO pin of the Raspberry Pi is used to trigger notification by pressing a button on the sensor device. The scheme used is as follows:

1. Push button and record time.
2. Receive notification and calculate difference.
3. Pick a random time to sleep and start new timer with it.
4. When the timer is fired go to step 1.

Measuring the delay from when a request is issued and data is received back is done by simply measuring the round-trip time (RTT). The Raspberry Pi request data from the SensorTag and measures the time it takes to receive data.

Both measuring techniques uses a random sleep interval between measurements. The reason behind this is that it generally cannot be known when an event occurs at the server side or when a client will request data. If a static time is used requests and notification may occur just before or after, or at any other time, an anchor point and thus skewing measurements. Randomly initiated measuring cycles better simulates a real world scenario where an event may occur at any time. The average latencies can be found in Table 5.1

Connection interval	RTT (ms)	Notification (ms)
7.5ms	23.84	47.19
25ms	46.93	63.21
50ms	84.13	55.96
100ms	161.80	82.45
500ms	765.24	285.5
1000ms	1534.88	532.31
2000ms	3201.43	1032.62

Table 5.1: Average latency for notifications and round-trip times. Notifications are one way transmissions whereas requests are back and forth and thus generally takes longer time.

Finally, there is one additional parameter that can have an effect on the latencies. The *connection slave latency* allows the slave device to skip anchor points. This will delay communication from the central device to the peripheral, but not vice versa. If the connection interval is set to 100ms and connection slave latency to 10, the slave will only wake up and listen to the central device every 1000ms. If notifying a value there will always be less than 100ms to the next anchor point.

5.2 Measuring power consumption

The TI SensorTag used throughout this thesis does not exhibit a constant current draw. Most of the time the device is in sleep mode consuming relatively little power. This behavior is illustrated in Figure 5.2 in which the device is set to advertise with a period of 1 second.

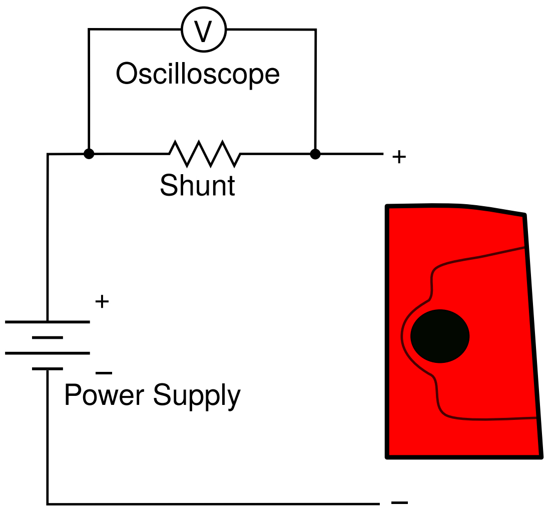

Figure 5.1: Current measurement using a resistor and oscilloscope.

Power consumption measurements have been conducted over a 10.2Ω shunt as recommended by Texas Instruments [24]. The resistor is connected in parallel with the positive terminal of the power supply as seen in Figure 5.1.

For measuring the voltage over the resistor an *Agilent InfiniiVision DSO-X 2024A* oscilloscope have been used.

5.2.1 Modifications

The SensorTag contains a DCDC converter (TPS62730 [23]), shortly mentioned in Section 3.3, used to decrease the power consumption of the device. The converter is controlled in the SensorTag software, usually by turning it on just before transmission window and disabled directly after. Some components on the SensorTag, such as the temperature sensor, are not compatible with the converter and in these cases the converter is not enabled. To make sure all measurements are done on the same basis the regulator has been permanently disabled, through software, in all the measurements conducted in this thesis.

For measuring, the standard 3V CR2032 battery is no longer used. Instead, two standard AA batteries in series, totaling 3.2V, are used. The reasoning behind this is that two standard AA batteries have a lower internal resistance and thus will affect the measurements less. Note the shunt on the positive terminal on the modified device in Figure 5.3.

5.2.2 Measurements

The radio in the CC2541 chipset is in sleep mode most of the time and wakes up only when processing sensor data and transmitting or receiving. This reduces total power consumption but makes it more difficult to measure it. It is required to measure the energy use during non sleep separately and because these may differ depending on what the SoC is processing all different types of wake ups must be isolated, measured, and used to calculate the actual consumption. For our uses the SensorTag will wake up for advertising in

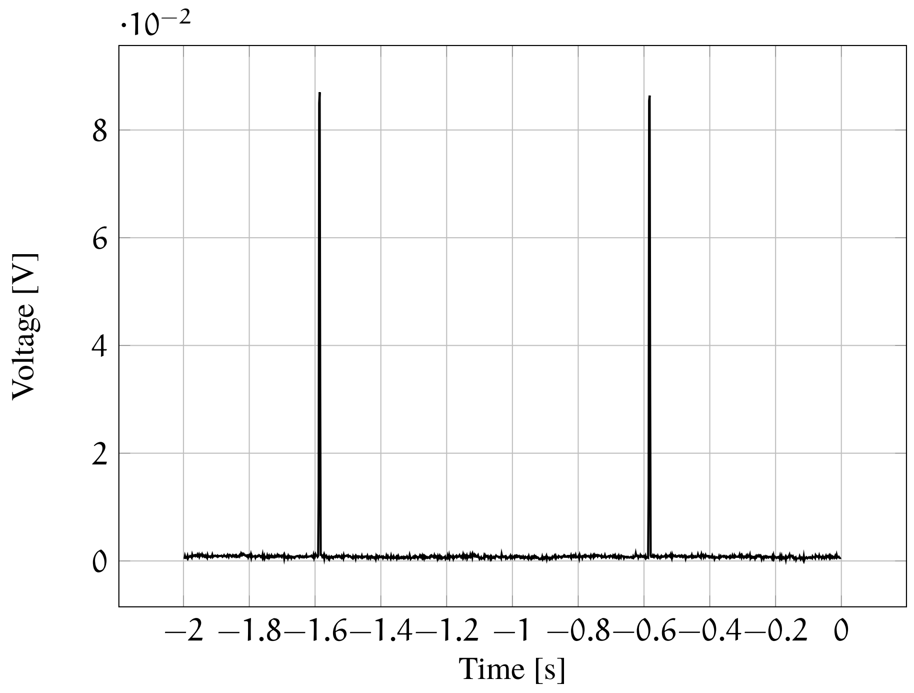

Figure 5.2: Two second capture of the SensorTag power consumption in Advertisement mode. It can be seen that the device is in sleep mode most of the time. The actual readings of the transmissions (the spikes) on the voltage axis is of less importance as the oscilloscope is set to measure over a relatively long period and thus the spikes cannot be measured properly.

advertisement mode, data transfer in connection mode, and for sensor sampling if a sensor is enabled. Captures of advertisement frames at a few different sizes can be seen in Figure A.2 in Appendix A.

In sleep mode current consumption is measured to 39.36μA in advertising mode and 39.76μA in connected mode.

5.2.3 Calculations

The oscilloscope readings need to be converted by using $U = R \cdot I$. The total charge of a frame in coulomb then becomes:

$$Q = \frac{1}{\Omega} \int_t \mathrm{V}(t)\, dt.$$

Ampere is defined as charge per second and thus the average ampere is $Q \cdot \frac{1}{T}$ where T is the period between frames.

Using this information together with the sleep mode current consumption from previous sections, as well as the period of transmissions, it is possible to create an estimation of the power consumption and battery longevity.

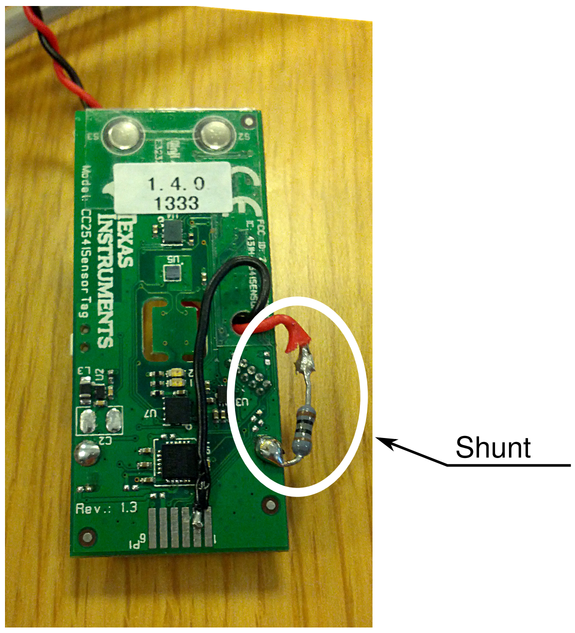

Figure 5.3: Modified SensorTag used in measurements.

5.2.4 Transmit Power effect on advertisement

Another potential way of reducing power consumption is to decrease the transmitter power output. The SensorTag SoC is able to transmit at 0dBm, −6dBm, and −23dBm. Transmitting at lower dBm will decrease the power output and thus lead to a lower power consumption, but it also affects transmission range negatively. It is interesting to know how much lower energy the consumption becomes when transmitting at a lower power and if this has any practical implications.

We conducted measurements of the power consumption while advertising three different sized payloads and compared these. The captured samples can be seen in A.2 and a summary of the three levels in Table 5.2. The summary specifies the power consumption of a single frame, as measured, at full power. By lowering power output it is possible to decrease the energy used slightly as indicated by the percentages.

N/Tx	0dBm	−6dbm	−23dBm
2	3.2300×10^{-5}C (100%)	99.36%	98.64%
15	3.7160×10^{-5}C (100%)	98.25%	97.93%
31	4.2490×10^{-5}C (100%)	98.47%	97.55%

Table 5.2: Current consumption and savings on advertisement frames with payloads of 2, 15 and 31 bytes. The first column show the energy used during advertisement. Decreasing the power output decreases power consumption as seen in the next two columns.

We also measured the effect transmitter power output has on range and packet loss at the three levels in Section 5.3.

5.3 Range and Packet loss

The distance between the SensorTag and the camera transceiver affects the reliability of the connection between them. Measurements were conducted at 0.5, 4, 8, 12, 16, 20, 24, 28, and 32 meters.

In Figure 5.4 the received signal strength indicator (RSSI) value reported by the BLE hardware is plotted against the distance. Only distances at which a link could be established and data could be retrieved are included.

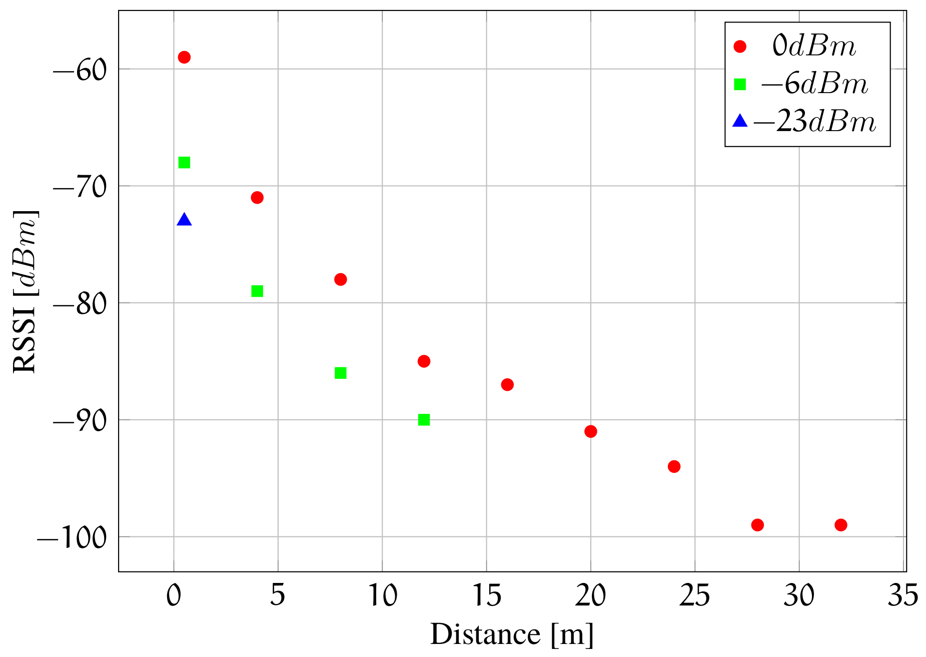

Figure 5.4: RSSI at different distances for the USB dongle used. Only distances where a connection could be established are included.

Since the Broadcom based USB radio used to communicate with the SensorTag is a controller (See Subsection 2.2.2) that only can be controlled using HCI, there is no easy way of detecting the amount of lost packages. For this reason a dedicated BLE sniffer is used in conjunction with the Broadcom radio. The sniffer software display malformed packets as well as the exact timing of the packets making calculating lost packages possible. If a packet is malformed with regards to its checksum, or if it is never seen at the receiving end, it is regarded as lost. The sniffer [8] reports slightly different RSSI values compared to the Broadcom radio used with BlueZ (See Figure 5.5). The average package loss at different distances can be seen below:

The distance measurements were conducted in an open office space with clear view between the SensorTag and the device acting as Central. It should be noted that a measurement similar to that in Table 5.3 was conducted outdoors away from any disturbances from other wireless devices and that resulted in a negligible packet loss at any of the distances.

	0dBm	−6dBm	−23dBm
0.5m	0%	0%	0%
4m	0%	0%	74.03%
8m	0%	0%	N/A
12m	0%	1.46%	N/A
16m	0.32%	0.82%	N/A
20m	0.14%	0.93%	N/A
24m	29.88%	36.84%	N/A
28m	36.20%	N/A	N/A
32m	90.37%	N/A	N/A

Table 5.3: Measurement of packet loss was conducted at different distances. N/A indicates no packets was received or that the packets received could not be decoded.

5.4 Using the SensorTag as a door opener detector

The results measured and found in previous sections is finally enough to evaluate the use of the SensorTag in a real world environment with regards to battery consumption.

The setup is as in Figure 1.1 in Chapter 1. The SensorTag is attached to a door and is configured to sample the gyroscope once per second. This is low enough to ensure it will detect a movement in the particular door type used which is a standard swinging door. To ensure a low latency from when sampling occur to when they reach the client in the camera a connection interval of 100ms is chosen. This means the time from when a reading is sampled to when there is an anchor point where it can be sent to the client is at most 100ms.

However, to keep the power consumption as low as possible it is required to maximize the period between anchor points, keeping the peripheral from waking up too often. To do this a connection slave latency of 20 is chosen. This way the SensorTag will not actually be required to wake up more than once every 2000ms. The combination of parameters results in the sensor waking up once every second, after every sample, to transmit the measurement. The average latency from a sampling until it was received by the client using these parameters was measured to be 81.1ms and the maximum latency was 946.34ms. This is well in line with the notification delay which was measured in Table 5.1 for a connection interval of 100ms, but does not strictly require any more power from the sensor device than an interval of 2000ms unless there is data to transmit. Only 1% of the samples had a latency of more than 200ms in the tests, and the estimated power consumption is given by:

- Sleep consumption of 39.76µA.
- One gyroscope sample per second. (313.85µA, Figure A.1)
- One gyroscope transmission per second. (23.8µA, Figure A.4)
- Possibly one empty connection frame every two seconds. (9.76µA, Figure A.3)

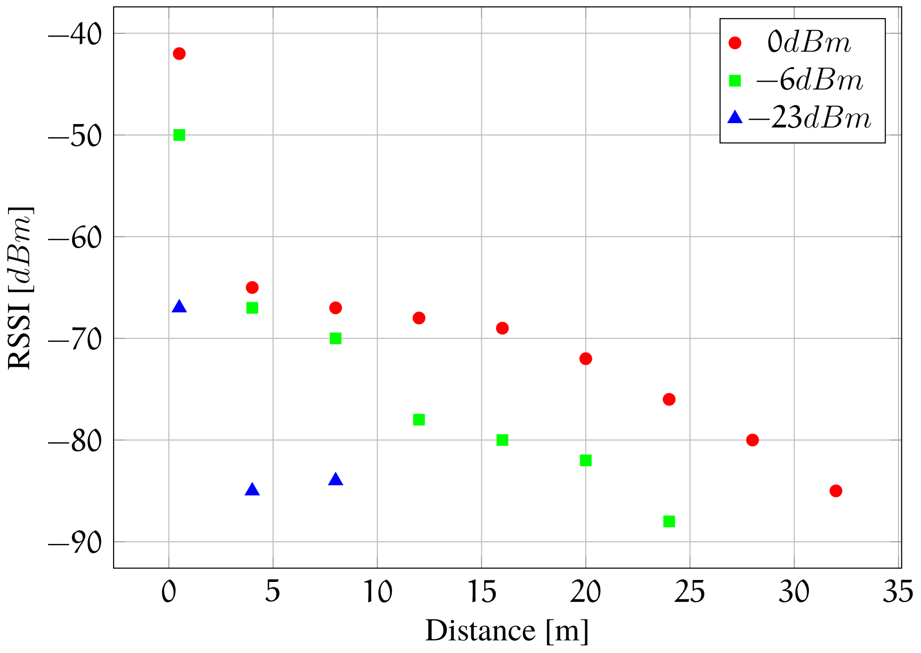

Figure 5.5: RSSI at different distances for the packet sniffer. Only distances where a connection could be established is included.

This gives an average consumption of 387.17μA. Using two standard AA batteries totaling about 4000 mAh would give an estimated run time of 430 days.

Chapter 6
Discussion

In Chapter 3 and 4 we developed a system for using wireless sensors using the Bluetooth Low Energy technology in an Axis camera environment. While a prototype was designed the results were deliberately left quite wide and not completely focused around the specifics of the door opener detector prototype. The reason behind this is to allow a discussion of the Bluetooth Low Energy standard for a wider range of sensor types. In this chapter we try to analyze the results and measurements made in the previous chapter to give an idea on the suitability of the standard.

6.1 Latency

In the higher connection intervals found in Table 5.1 it can be seen that polling a value, that is asking for it and getting an answer, is around three times higher compared to when the peripheral decides to use a notification. It may seem strange that going back and forth should take more than twice as long time as simply sending a one way message. The answer is that when a request is to be made the central first waits until an anchor point to make the request. In average this takes half a connection interval. The peripheral registers the command but will not answer until the next anchor point, giving a delay of around 1.5 times the connection interval. A notification will only take around a half connection interval until it can be processed by the client.

Smaller connection intervals does not display the same relationship between connection interval and latency and the reason likely is that there is a significant delay in the software, both the testing code and the rest of the solution. On top of this the wireless transfer takes time and the radios need some time to process the data.

In the very smallest connection intervals notifications take significantly longer time than requests which is very strange. The exact reason behind this was not found out, but we suspect this to be due to notifications being handled slightly different compared to requests in the client software. When a client requests data using the D-Bus interface it blocks until an answer is received. Notifications on the other hand are treated as something that may occur at any time. After the Raspberry Pi triggers the SensorTag, the client will enter the GLib main event loop. Here it may do some other tasks or it may enter sleep mode until it decides a GLib event is ready to be processed.

In Section 5.4 the *connection slave latency* were used to minimize the latency from peripheral to central. A connection interval of 100ms ensures the central is available to receive a notification every 100ms but since the peripheral only need to wake up every 20 anchor points or every 2000ms the central to peripheral latency becomes much higher. Setting a high connection slave interval works well in cases where the peripheral where the latency from when an event occurs to when it should reach the central but is detrimental if the peripheral must act quickly on requests and commands from the central. In our case, where the only commands issued to the peripheral are to enable and disable sensors and notifications, it makes sense to use it as the peripheral only need to wake up to send samples and not in between which saves power. However, there cannot be a too high connection interval, Apple for example dictates that a frame must be sent at least once every 2 seconds in their guidelines [19]. This is because a timing between central and peripheral must be maintained. Every time a frame is sent back and forth the devices are synchronized and the longer time between frames are the more exact the timing device must be. BLE can use cheap oscillators to keep track of the timings.

6.2 Power consumption

From our measurements in Figure A.2 and A.4 in Appendix A, it can be concluded that connection frames use less power than advertising frames with the same amount of data. The reason behind this is that advertisements are done over three channels at once. Connection frames are only sent on one channel per anchor point and thus uses less power.

The power consumption in sleep mode was surprisingly high compared to what is stated in the data sheet for the radio [22]. In the datasheet the figures of 1µA and 0.5µA is stated while we measured 39.36µA and 39.76µA. In both cases the difference in consumption between the power modes used in advertising mode and connected mode differ by around 0.5µA which indicates that the correct power modes are used in the SoC. It is likely that some components on the SensorTag use a relatively large amount of power that may be optimized to further reduce power consumption.

Reducing the power output of the radio can help to reduce the power consumption slightly, but as seen in Table 5.2 a saving of slightly more than 2% was possible in the cases with larger payloads and not even that with the smallest payload. But it was very detrimental to the range (See Section 5.3). 2% might seem enough to warrant further consideration but given the fact that data transmission in our use case only accounts for less than 9% of the total power consumption and means much worse range it was not considered further. The size of the data payload on the other hand had a larger effect. The maximum payload used around 30% more energy compared to the smallest.

6.3 Range

As stated in the previous section, lowering the power output will worsen range while not improving the power consumption significantly in the setup used. Using a lower power output could be useful if a shorter range is desired. At $-23dBm$ it was only usable at very close range. In our case this loss of range was not desired.

Table 5.3 shows that the solution worked fine for distances at 20 meters and below but more than that led to high packet losses. The environment where the measurements were made was close to other standard IEEE 802.11 wireless systems and it is likely that they interfered. It should also be noted that the packet loss over 20 meters often occurred in batches which indicates the link was being disturbed by other 2.4GHz devices as well. When similar measurements were conducted outdoors away from potential disturbances the packet loss dropped to almost 0.

A problem during range testing was that it was not possible to use the BCM20702 based radio, the one used to actually communicate with the sensor device, as it was hard to get any information on the packets sent through the air. The packet sniffer used a TI CC2540 radio and was capable of picking up packets fairly consistently without line of sight in a much better way than the BCM20702. This was probably due to the differences in antenna. The TI CC2540 packet sniffer had an etched antenna much larger than that of the BCM20702.

6.4 Connection schemes for minimizing power consumption

From the results in Chapter 5 we can conclude that power consumption, latency and range depend on each other. Power consumption depends on how often the sensor has to wake up from sleep. We can further conclude that BLE is very flexible and allows many connection schemes and picking the correct one allows for a minimal power consumption. If a low latency is required and it is critical for data to reach the client, and that we can detect a loss of connection such, as in our door opener detector case, a connection must be kept open. If the low latency is only required from peripheral to central and not vice versa, the connection slave latency can be increased which reduces the amount of times a peripheral have to wake up and a low latency can be achieved without increased energy use. On the other hand if a low latency is required both between central and peripheral power consumption will increase as can be concluded from Subsection 5.2.3.

Some types of sensors might not need a connection running. For example a room temperature sensor could simply broadcast the readings in the advertisement data which might be picked up by any device interested in the data. However, advertisement frames are larger as they have to be transmitted over three channels simultaneously and it is important to make sure as little data as possible is advertised.

A third type of sensors that detects occasional non latency critical events could simply stay offline until an event has occurred at which time it starts advertising and wait for someone to connect. The connecting device could then retrieve whatever event occurred and clear it, allowing the sensor to go back to sleep. A downside to this approach is it makes detecting a broken link harder. During connection if the link is disconnected it is immediately discovered when the next expected packet is not received. This is not the case if the sensor sleeps for a non deterministic amount of time and then tries to advertise itself.

6.5 Bluetooth Low Energy for wireless sensors

Our system was only reliable up to about 20 meters indoors with line of sight. This is not very good for a solution where the placement of the sensors is not flexible. A door opener detector must be placed on the door in question and the placement of receiving cameras is not necessarily flexible either. But this is not enough to draw any definitive conclusions on using BLE on these kinds of systems because there are other radios capable of power output of up to +10dBm compared to the 0dBm used by our sensor. Also, it is possible to use larger external antennas instead of the small ones used in our transmitters. When doing less formal comparisons of the BCM20702 and the CC2540 we saw that a better match of transmitter and receiver most likely would increase range, even without increasing power output, sadly the CC2540 packet sniffer could not be used directly with BlueZ without very extensive modifications to the firmware. For an even larger range it would be possible to add a range extending unit which is used for amplifying the signal from the radio[11]. Such a component however would affect the power consumption adversely.

We experienced a large packet loss over 20 meters, probably due to interference, but this does not necessarily mean the system is unusable at the range. If the use case does not require data to be transmitted at a timely manner it would still be quite usable. If for example some status report, like temperature or humidity is required once every hour the central could simply retry until it is able to get the data. Our use case on the other hand requires a fairly low latency from sampling for the camera to be able to act.

When designing wireless sensors power consumption is a very important aspect, but as we can see in our measurements in Section 5.4 the radio is not necessarily a limiting factor. The gyroscope sensor stands for 81% of total power consumption where the transmissions only stand for 9%. Instead of using the gyroscope we could have used some mechanical connector. This would have eliminated the 337µA used by the gyroscope sensor while still achieving the same goal. The average consumption could then be brought down to around 50µA plus some more for notifications that only had to be sent whenever the mechanical connector is triggered. This setup could theoretically last for as long as 9 years on 4000 mAh and be almost exclusively be limited by the BLE chipset.

6.5.1 ZigBee

It is not possible to make a fair comparison between technologies such as Bluetooth Low Energy and ZigBee without creating equivalent hardware solutions and optimize them for that particular use case individually. Even so, we can say some words on them. In the oscilloscope captures in Figure A.2 in Appendix A we can see that the current consumption during transmission lies around 15mA during transmission. The TI CC2530 ZigBee enabled SoC, which is roughly comparable to the CC2541 used in our measurements, uses about 30mA and according to [10] this is normal for ZigBee radios. In [34] it is stated that BLE has an energy utilization around 2.5 times better than ZigBee.

Even though it can be concluded that BLE is more energy efficient than ZigBee that does not necessarily have a significant impact as data transfer only accounted for around 12% of the SensorTag's energy consumption.

Range wise the two technologies are comparable being specified from 10 meters up to 100 meters. ZigBee can operate at 915MHz and 868MHz as well as 2.4GHz which would allow a larger range, but less data throughput [39].

ZigBee makes the equivalent of the central stay awake and listen for packets all the time. In BLE data transfer can only occur at the specific anchor points. When data transfer only occur at specific anchor points it is possible to decrease power consumption of the receiver as it does not need to listen constantly. On the other hand the peripheral device need to be available during these prenegotiated anchor points. Thus, this difference may work to advantage or a disadvantage depending on the use case.

Another potential advantage Bluetooth Low Energy has over ZigBee is that most new mobile phones have support for it and many Bluetooth 4.0 modules have support for both LE and BR/EDR using the so called Dual Mode radios. Because many devices automatically have support for Bluetooth LE, essentially for free, consumers might favor it over ZigBee although this is pure speculation.

6.5.2 Bluetooth BR/EDR comparison

The new standard of Bluetooth Low Energy and the older Bluetooth BR/EDR may share name but they are not comparable in other ways than that it can share some components to make dual mode radios possible.

Bluetooth BR/EDR was designed to be able to stream relatively large amounts of low latency data such as audio. It is hard to find good power consumption figures but in [38] figures of 10mA and 20mA which would be continuous during streaming. Bluetooth LE on the other hand is designed specifically for communicating *state* which equals small amounts of data and is not at all capable of streaming.

Furthermore, the method of discovery and connection differs. In Bluetooth BR/EDR devices use *inquiring* which means when they want to find other devices they send out a packet. If a device wants to be discovered it will have to listen for inquiries and respond to them. The opposite is true in Bluetooth LE. If a device wants to be discovered or connected to, it has to send out advertisements on the specific advertisement channels which are used to initiate a connection [16].

6.6 Better Security

In our solution we opted to use paired mode with the *Just Works* key exchange scheme. In Subsection 2.2.6 we briefly noted that the Security of the BLE standard was not enough to protect against a passive eavesdropper. If security is strictly required it is still possible to achieve a satisfactory level of security by simply using application level encryption and key exchange. This is further simplified by the fact that the interface for utilizing the hardware crypto engines are public and very much usable from application level in both the CC254X and nrf51822 SoCs that we studied.

A second approach would be to distribute keys manually by programming the sensors during installation and using the out-of-band approach for the Security Manager. It would even be possible to use a custom GATT service to exchange keys. By implementing a service enabling the exchange of shared secret using Elliptic Curve Diffie Hellman (ECDH) and using that shared secret as a temporary key there would be no known way of breaking it by a passive eavesdropper.

The scheme would look roughly like this:

1. Both parties generate a public and private key pair.
2. The master queries the peripherals ECDH service and supplies its public key.
3. The peripheral responds with a public key.
4. Master and slave uses exchanged data to generate temporary key which is fed to the Security Manager using the out-of-band capability.

Optionally there could also be a step where the two devices are verified, such as by a common and random pin code, to make it much stronger against an active MITM attack.

First and foremost a method of generating a ECC key pair and conduct a generation of a shared secret is required. We shortly tested an open source library named *micro-ecc* [28] on a *Nordic Semiconductor nRF51822 Evaluation Kit* with some success, however implementing a whole solution with a GATT service required too much work for this thesis. A simulated key exchange without any over the air transfer worked well which indicates that with the proper services secure Bluetooth Low Energy communication could be realized today for custom use cases.

Chapter 7

Conclusions

At the end of this thesis project we have succeeded in integrating a Bluetooth Low Energy sensor into an Axis camera. The sensor is capable of transmitting sensor data to a camera which is in turn able to act upon these readings, for example by pointing the camera at a preset direction or starting a recording. After tuning the parameters of the sensor and replacing the original CR2032 coin cell battery with two standard AA batteries a satisfactory battery life was achieved.

Sadly our system was only reliable up to about 20 meters indoors with line of sight. We are using sensors to detect if a door opens and allow a camera to detect this. The optimal result would be one where the positioning of both sensors and receiver does not have to be considered. A door opener detector must be placed on the door in question and the placement of receiving cameras are typically quite constrained as it is not possible to place a camera at any location. The camera should be placed where it has the best coverage and not be limited by the placement of a wireless sensor.

This is not enough to draw any definitive conclusions on using BLE on these kinds of systems however as there are radios capable of higher power output. It is also possible to use larger antennas in both sensor and camera. As a last resort there are external range extending amplifiers available for increasing power output even more, with a higher power consumption as a result.

In order to integrate a sensor into a camera a few steps have had to be taken. First BLE services for the sensors available in the sensor development kit was implemented and added to BlueZ. Secondly a software for interfacing with BlueZ were developed as an app using the Camera Application Platform. Thirdly the sensor where used as a door opener detector and the transmission parameters were tuned to this particular use case and measurements were made.

7.1 The future

During this thesis a Raspberry Pi was used as a proxy between a camera and a commercially available radio. In order to actually integrate this into a camera some hardware modifications has to be made. For example, a radio unit has to be physically integrated. Even so, after hardware modifications the software used and developed in this thesis should still be usable. It would be a matter of simply by moving it into the camera as standard IPC D-Bus has been used, with the exception that in this thesis messages were sent over TCP/IP instead of locally.

Bluetooth Low Energy support was at the start of this thesis not very widely implemented. Certain platforms such as iOS and Android have good support for peripherals. When it comes to Linux it seems as though BlueZ is under heavy development at the moment. During this thesis much has happened in BlueZ to enable and simplify more BLE functionality. Very recently there have been patches to allow BlueZ to export GATT services as well as use external ones and thus allow BlueZ to act as a peripheral.

Also, development on a proper D-Bus interface for implementing GATT services, both remote and local, has begun which will greatly simplify adding support for new services as modifications to the BlueZ code base itself will not be necessary. Typically, the profiles implemented in Chapter 4 would use this interface [36].

Bluetooth 4.1 was released in December 2013 and it brings some interesting features. Among the most important is the addition of *L2CAP Dedicated Channels* which could be used to make IPv6 enabled Bluetooth Low Energy sensors [37]. This could somewhat mitigate the lack of mesh networking which is one of the advantages of ZigBee. If IPv6 support is enabled it is not far fetched to imagine BLE devices advertising *I want to send data* and so called *gateways* connecting to these devices and further route the packets. A cell phone, or a fixed network connected surveillance camera, might well double as a gateway.

Furthermore, it is well worth investigating other standards like ZigBee. As we stated earlier in the paper, depending on the actual sensors used, the radio might be a relatively small contribution to the total power consumption. And there exist similar radio chipsets with integrated radio for other standards as well. There are several proprietary standards available and it is also quite possible to create new protocols on top of IEEE 802.15.4. A protocol customized for very specific uses will arguably result in the lowest power consumption. Relying on a commonly used standard on the other hand will potentially result in a greater interoperability as the designers of the standards have put a lot of thought into them.

Lastly, this thesis has focused on using BLE in order to let a camera use wireless sensors. Considering that most modern smartphones, at least newly released, has support for BLE it would be interesting to investigate if and how BLE could be used for transmitting data from for example a camera to a mobile phone. In this setup BLE wouldn't be considered because of the low power consumption but rather because of its widespread support which might allow for easy interfacing.

Glossary

ATT Attribute Protocol.

BLE Bluetooth Low Energy.

BR/EDR Basic Rate and Enhanced Data Rate.

ECDH Elliptic Curve Diffie Hellman.

GAP Generic Access Profile.

GATT Generic Attribute Profile.

GFSK Gaussian Frequency Shift Keying.

HCI Host Controller Interface.

IPC interprocess communication.

ISM industrial, scientific and medical.

L2CAP Logical Link Control and Adaptation Protocol.

MITM man-in-the-middle.

RSSI received signal strength indicator.

RTT round-trip time.

SM Security Manager.

SoC System on a Chip.

UUID universally unique identifier.

Bibliography

[1] *A Look at the Basics of Bluetooth Technology*. URL: `http://www.bluetooth.com/Pages/How-it-Works.aspx` (visited on 03/10/2014).

[2] *A Portable User-Space Bluetooth Stack*. URL: `https://code.google.com/p/btstack/` (visited on 09/16/2013).

[3] Axis Communications AB. *AXIS Q6034 PTZ Dome Network Camera*. 2012. URL: `http://www.axis.com/files/datasheet/ds_q6034_42102_en_1211_lo.pdf` (visited on 02/10/2014).

[4] Axis Communications AB. *Event and Action Services*. 2013. URL: `http://www.axis.com/files/manuals/vapix_event_action_51386_en_1307.pdf` (visited on 02/21/2014).

[5] Mai Ali, Lutfi Albasha, and Hasan Al-Nashash. "A Bluetooth low energy implantable glucose monitoring system". In: *Microwave Conference (EuMC), 2011 41st European*. IEEE. 2011, pp. 1265–1268.

[6] *BlueZ README*. 2014. URL: `https://git.kernel.org/cgit/bluetooth/bluez.git/tree/README?id=93d07e94429bc367afc061a32d4dc71ee5a94c39` (visited on 02/20/2014).

[7] SIG Bluetooth. *Specification of the Bluetooth System-Version 4.0*. 2010.

[8] *CC2540 USB Evaluation Module Kit*. URL: `http://www.ti.com/tool/cc2540emk-usb` (visited on 02/21/2014).

[9] *CC2541 SensorTag Development Kit*. URL: `http://www.ti.com/tool/cc2541dk-sensor` (visited on 12/05/2013).

[10] Eduardo Casilari, Jose M Cano-García, and Gonzalo Campos-Garrido. "Modeling of current consumption in 802.15. 4/ZigBee sensor motes". In: *Sensors* 10.6 (2010), pp. 5443–5468.

[11] Abhishek Chattopadhyay. *Using CC2590 Front End with CC2541*. 2013. URL: `http://www.ti.com/lit/an/swra422/swra422.pdf` (visited on 03/20/2014).

[12] *Coexistence between ZigBee and Other 2.4GHz Products*. 2013. URL: `http://www.atmel.com/Images/Atmel-42190-Coexistence-between-ZigBee-and-Other-24GHz-Products_AP-Note_AT02845.pdf` (visited on 03/03/2014).

[13] Colin Walters David Wheeler John Palmieri. *D-Bus Tutorial*. Red Hat, Inc. URL: `http://dbus.freedesktop.org/doc/dbus-tutorial.html` (visited on 02/20/2014).

[14] Joe Decuir. "Bluetooth 4.0: Low Energy". In: *Cambridge, UK: Cambridge Silicon Radio SR plc* (2010).

[15] Luiz von Dentz. "BlueZ - Plugging the Unpluggable". In: Tizen. 2012. URL: `http://download.tizen.org/misc/media/conference2012/wednesday/bayview/2012-05-09-0900-0940-bluez-_plugging_the_unpluggable.pdf` (visited on 02/19/2014).

[16] N. Gupta. *Inside Bluetooth Low Energy*. Artech House Remote Sensing Library. Artech House, 2013. ISBN: 9781608075799.

[17] R. Heydon. *Bluetooth Low Energy: The Developer's Handbook*. Pearson Education, 2012. ISBN: 9780132888400.

[18] Marcel Holtmann. *[PATCH 1/2] Bluetooth: Enable Low Energy support by default*. May 3, 2012. URL: `http://www.spinics.net/lists/linux-bluetooth/msg24480.html` (visited on 02/11/2014).

[19] Apple Inc. *Bluetooth Accessory Design Guidelines for Apple Products*. Sept. 18, 2013. URL: `https://developer.apple.com/hardwaredrivers/bluetoothdesignguidelines.pdf` (visited on 03/10/2014).

[20] Apple Inc. *Core Bluetooth Programming Guide*. 2013. URL: `https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html` (visited on 02/20/2014).

[21] Google Inc. *Bluetooth Low Energy | Android Developers*. 2014. URL: `https://developer.android.com/guide/topics/connectivity/bluetooth-le.html` (visited on 02/20/2014).

[22] Texas Instruments. *2.4-GHz Bluetooth low energy and Proprietary System-on-Chip*. 2013. URL: `http://www.ti.com/lit/ds/symlink/cc2541.pdf` (visited on 03/10/2014).

[23] Texas Instruments. *Step Down Converter with Bypass Mode for Ultra Low Power Wireless Applications*. 2012. URL: `http://www.ti.com/lit/gpn/tps62730` (visited on 02/11/2014).

[24] Sandeep Kamath and Joakim Lindh. "Measuring bluetooth low energy power consumption". In: *Texas instruments application note AN092, Dallas* (2010).

[25] Maxim Krasnyansky. *BlueZ: Official linux bluetooth protocol stack*. 2003.

[26] Jimmy Assarsson Linus Karlsson. "Performance of constrained wireless devices in the Internet of Things". In: (2013). URL: `http://sam.cs.lth.se/ExjobGetFile?id=542` (visited on 05/14/2014).

[27] Matthew Loy, Raju Karingattil, and Louis Williams. “ISM-band and short range device regulatory compliance overview”. In: *Texas Instruments, Application Report SWRA048* (2005). URL: `http://www.ti.com/lit/an/swra048/swra048.pdf` (visited on 02/25/2014).

[28] Ken MacKay. *A small ECDH implementation for 32-bit microcontrollers*. URL: `https://github.com/kmackay/micro-ecc` (visited on 03/02/2014).

[29] Elke Mackensen, Matthias Lai, and Thomas M Wendt. “Bluetooth Low Energy (BLE) based wireless sensors”. In: *Sensors, 2012 IEEE*. IEEE. 2012, pp. 1–4.

[30] *Official Linux Bluetooth protocol stack*. URL: `http://www.bluez.org/` (visited on 09/16/2013).

[31] *Raspberry Pi official website*. URL: `http://www.raspberrypi.org/` (visited on 09/16/2013).

[32] Mike Ryan. *How Smart is Bluetooth Smart?* 2013.

[33] Bluetooth SIG. *Device Information Service*. May 24, 2011. URL: `https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=238689` (visited on 02/19/2014).

[34] Matti Siekkinen et al. “How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15. 4”. In: *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*. IEEE. 2012, pp. 232–237.

[35] *Skyrocketing demand for Bluetooth appcessories for latest phones*. Bluetooth Special Interest Group. 2013. URL: `http://www.bluetooth.com/Pages/Mobile-Telephony-Market.aspx` (visited on 02/23/2014).

[36] Claudio Takahasi. *[PATCH BlueZ v2 01/18] doc: Add GATT API*. Feb. 21, 2014. URL: `http://www.spinics.net/lists/linux-bluetooth/msg42972.html` (visited on 03/11/2014).

[37] *Updated Bluetooth® 4.1 Extends the Foundation of Bluetooth Technology for the Internet of Things*. 2013. URL: `http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=197` (visited on 03/11/2014).

[38] David Weber et al. “A single-chip CMOS radio SoC for v2. 1 Bluetooth applications”. In: *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*. IEEE. 2008, pp. 364–620.

[39] *ZigBee Specification Overview*. 2014. URL: `http://www.zigbee.org/Specifications/ZigBee/Overview.aspx` (visited on 03/10/2014).

Appendices

Appendix A
Oscilloscope captures

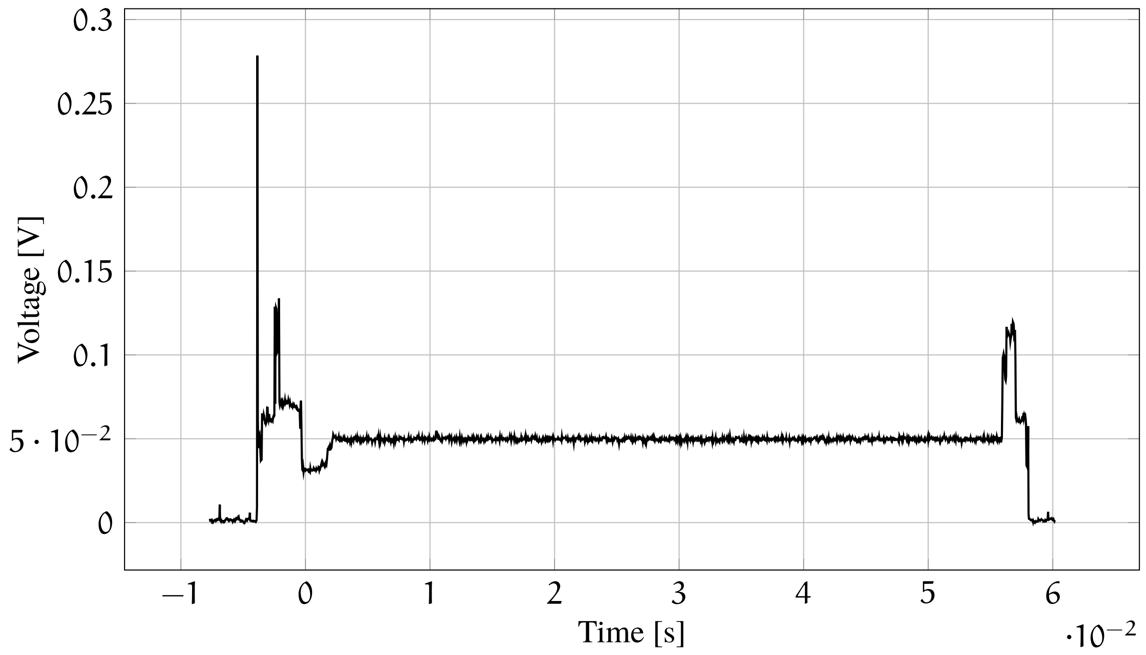

Figure A.1: Capture of a gyroscope sampling.

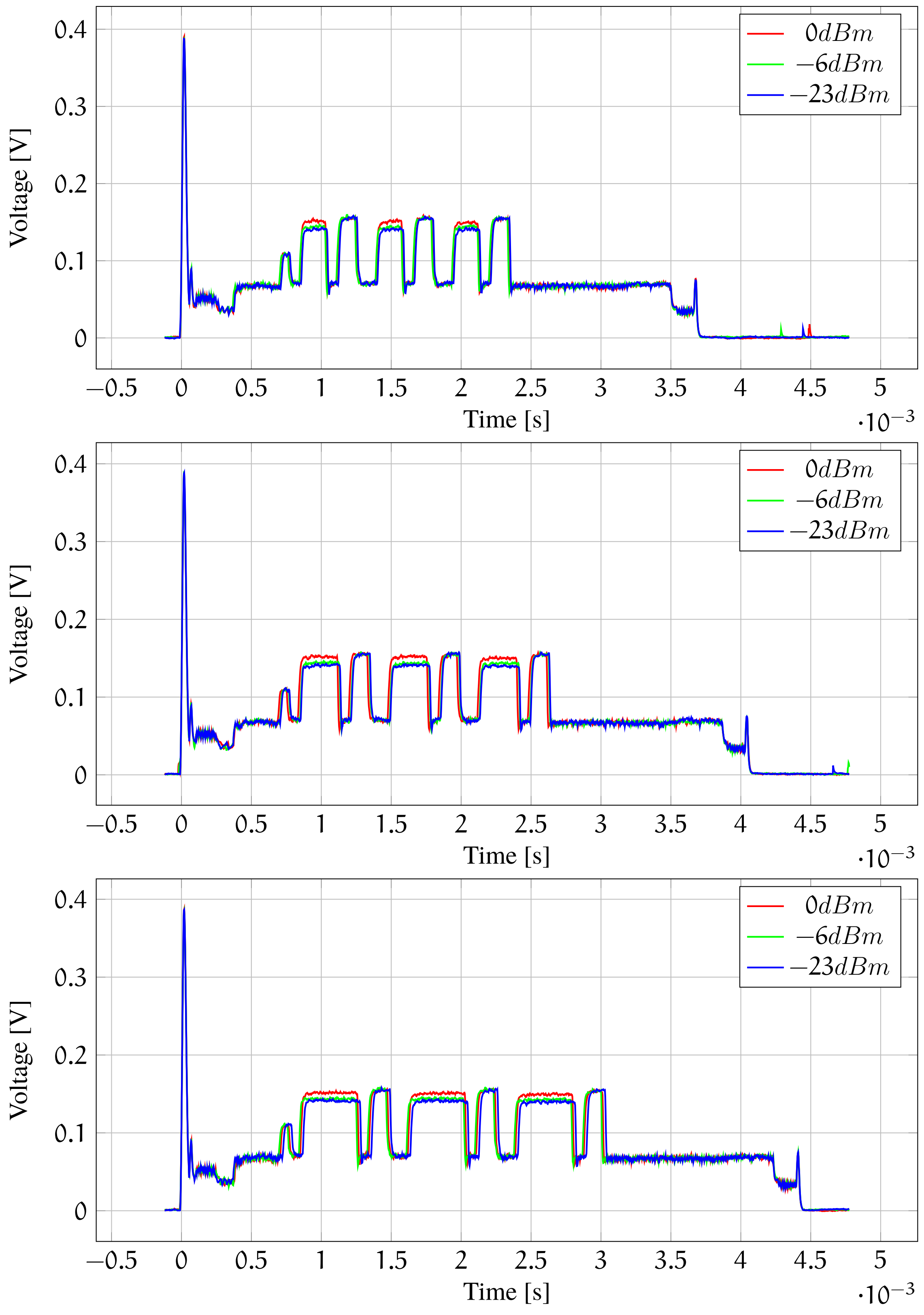

Figure A.2: Capture of advertisement frames with payloads at sizes 2, 15, 31 bytes respectively.

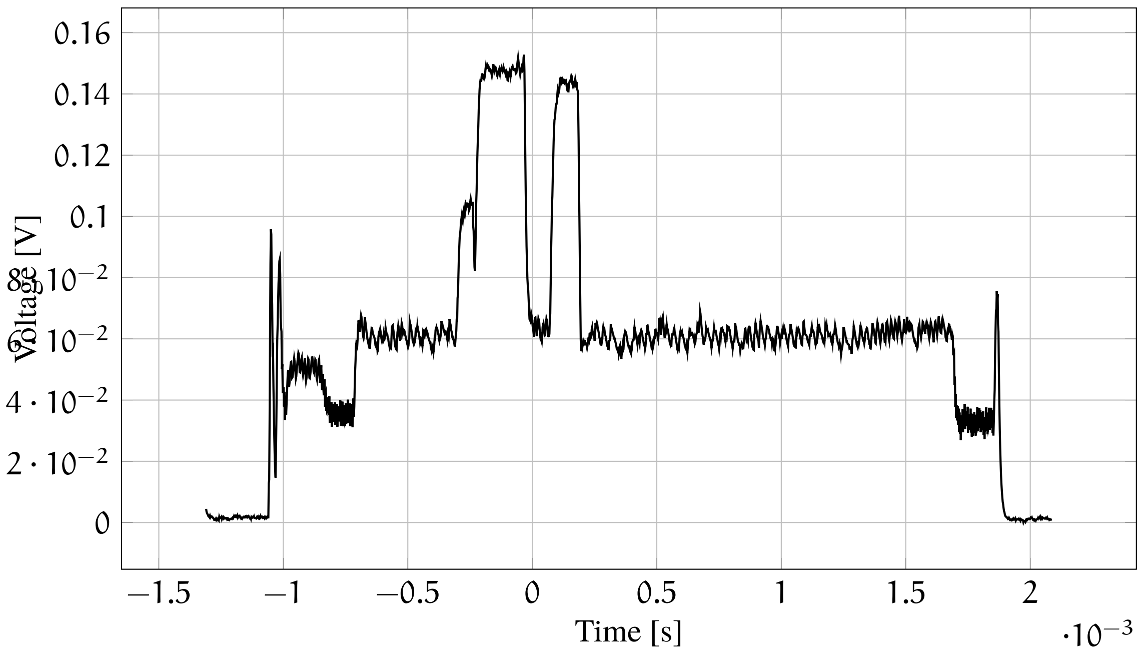

Figure A.3: Capture of a empty connection frame.

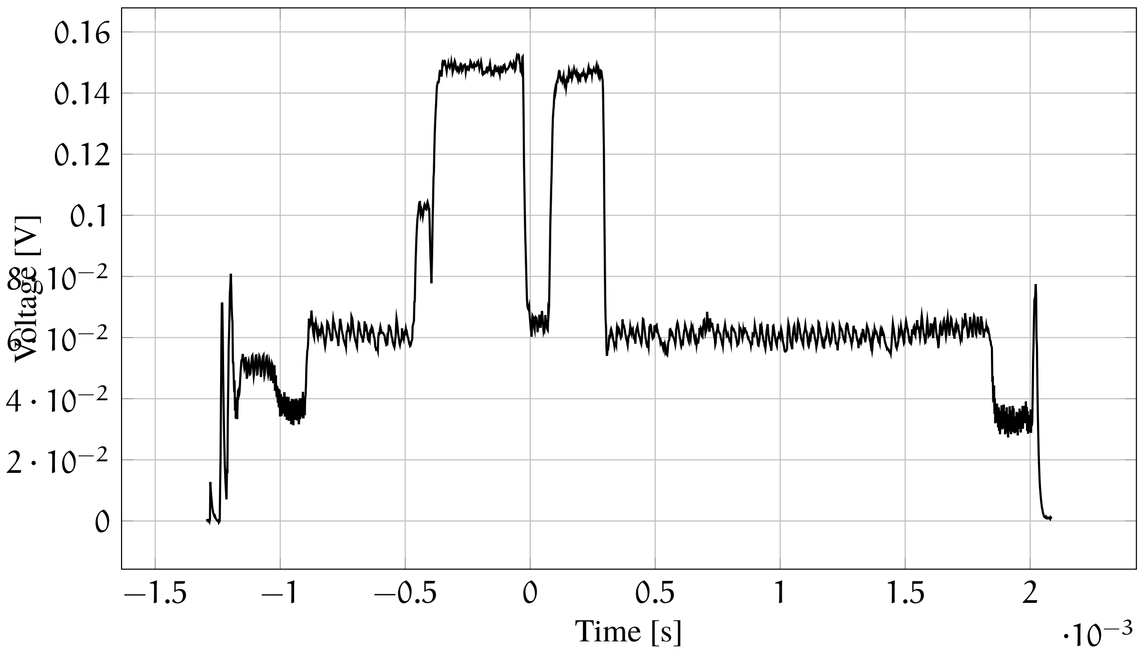

Figure A.4: Capture of a notification with 6 bytes of gyroscope data.

Appendix B
Misc

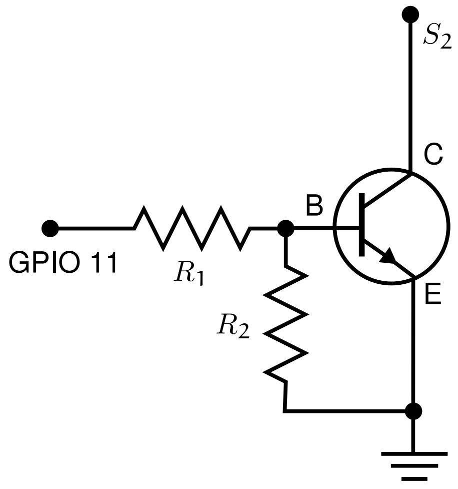

Figure B.1: S_2 is the positive side of the a button on the SensorTag and GPIO 11 a pin on the Raspberry Pi. When GPIO 11 is HIGH, current flow from S_2 to GND (equivalent to a pushed button), when its LOW no current flow (equivalent to not pushed button)

Bluetooth Low Energy in an embedded system

POPULAR SUMMARY
JOHAN ANDERHOLM

Supervisor: Jimmy Rubin (AXIS)
Examiner: Flavius Gruian (LTH)

Introduction

Our surroundings are full of data that can be a very useful input in a wide range of applications if only it can be gathered properly. It may be tempting to perform heavy image processing using the camera in a mobile phone or a network camera to accomplish relatively simple operations such as detecting whether someone enters through a door. However a well placed and specialized door monitor sensor is often both more reliable and less resource consuming while not necessarily being more complex than a mechanical switch. But with sensors comes the problem of interfacing. Cables often requires elaborate installation as well as physical connectors on the devices. The latter might be very hard to achieve in certain embedded systems, and is often not extensible without completely redesigning the hardware. Wirelessly connecting sensors can mitigate these issues and sometimes allow very simple installation in, but at the same time they come with other issues. Low energy consumption becomes very important in order for sensors to operate continuously on small battery packs, and sufficient range is also required for them to have an advantage over wired ones. A standard designed for energy efficient wireless sensors is Bluetooth Low Energy (BLE), part of Bluetooth 4.0, and this thesis investigates how it can be used in a Linux based embedded system. A BLE development kit is used as a door monitor and integrated into the event system of a camera. The performance is then evaluated in terms of power usage and range.

Method

The thesis was conducted at Axis Communications. In order to integrate the door monitor into a camera, the camera first needed a BLE capable radio. Integrating a radio into an existing camera is in itself a large task and outside the scope of this thesis. Instead an existing USB connected radio was attached to a Raspberry Pi which acted as a proxy so that the camera could communicate with the radio and, in extension, the door monitor sensor. This way focus could be directed towards the software. The BLE development kit contains several sensors, among others a gyroscope and some switches. The choice of sensors obviously have a large impact on power consumption, a gyroscope that has to be sampled requires more power than a simple switch. To cover a broader variety of sensor types two ways of detecting if a door is opened was trialed. One where a simple mechanical switch is used, and one where movements are detected using the gyroscope.

Software Architecture

Bluetooth support in Linux is divided into the kernel drivers and BlueZ which is a userspace Bluetooth stack. Applications using wireless devices are communicating with the stack through an IPC system called D-Bus. BlueZ was placed on the Raspberry Pi and the applications for controlling BlueZ, that is performing tasks such as instructing BlueZ to scan for devices or commence pairing and retrieving data from the wireless sensors, was developed and placed on a camera. Whenever motion is detected an event is dispatched which in turn can be handled in any fitting way by the rest cameras firmware.

BLE uses the notion of services to define the interface for using functions of a device. There are several standardized services but the development kit used vendor specific services that had to be manually implemented into BlueZ before the sensors where usable.

The standard of BLE allows for careful calibration of communication parameters in order to minimize power consumption. While sleeping the power consumption was about 40 μA compared to a consumption of around 15 mA during transmission of data. The ability to wake up no more than necessary to perform samplings and transmissions and thereby maximizing the time asleep was an essential step in reducing the power consumption.

Result

The thesis resulted in a fully functioning prototype with acceptable power consumption, but range wise there is much lacking for this particular use case. Using a gyroscope sampling rate of one second, enough to detect when a door opens, the estimated run time on two alkaline AA is around 430 days. If using a mechanical connector the run time could in theory reach nine years, although it would most likely be limited by the shelf life of the batteries used. This demonstrates the impact of sensor choice when detecting a certain happening. It also shows that the wireless transmission is not necessarily the main power consumer in simple sensors.

To much disappointment the prototype was only capable of communicating information reliably up to a mere 20 meters, making it a poor choice for this specific type of sensors. But with better antennas and a higher transmitter power output this could likely be increased at the cost of increased power consumption.