Internet of Things: Managing Deployments of BLE Nodes

Kevin Bluett
B.A.(Mod.) Computer Science
Final Year Project April 2015
Supervisor: Dr. Jonathan Dukes

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
Internet of Things: Managing Deployments of BLE Nodes

Supervisor: Jonathan Dukes

Kevin Bluett
School of Computer Science & Statistics
University of Dublin

This dissertation is submitted for the degree of
Bachelor of Arts (Moderatorship)

Trinity College April 2015
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

Kevin Bluett
April 2015
Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Kevin Bluett
April 2015
Acknowledgements

Firstly, thank you to my supervisor, Dr. Jonathan Dukes, for his unbounded help, advice, and support throughout the project.

Thank you to my family, and Sophie, for their constant love, patience, and support.
Abstract

An increasing number of embedded devices are being introduced into physical environments, this movement is known as the "Internet of Things". Embedded devices traditionally have limited methods for deploying software meaning that developers must be in the field to perform deployments. These existing solutions to this problem fall short.

This paper proposes a solution to overcome this problem. A management mesh network utilising a new protocol for IoT nodes. The result is a fully operational infrastructure for the deployment and management of Nordic nRF51 devices. The key feature is that it allows for the update of nodes over multiple hops.
# Table of contents

List of figures | ix  
List of tables | x  
Nomenclature | x  

## 1 Introduction  

## 2 Background  

### 2.1 The Internet of Things  
#### 2.1.1 Uses of IoT Technologies  

### 2.2 Technologies  
#### 2.2.1 Developing for Embedded Systems  
#### 2.2.2 Deploying to embedded devices  
#### 2.2.3 Limitations of existing tools  
#### 2.2.4 Devices  
#### 2.2.5 Nordic Semiconductor nRF51  
#### 2.2.6 Bluetooth Low Energy (BLE)  
#### 2.2.7 Protocols for the IoT  

### 2.3 State of the Art  
#### 2.3.1 TinyOS  
#### 2.3.2 Contiki  
#### 2.3.3 Similar Work  

## 3 Design & Implementation  

### 3.1 Requirements  

### 3.2 Developing for the IoT  

### 3.3 Architecture of IoT  

### 3.4 Structure of Solution  
#### 3.4.1 Edge Technology  
#### 3.4.2 Access Gateway
| 3.4.3 Application Layer | ........................................ 21 |
| 3.5 Hardware | ........................................ 21 |
| 3.6 Language & Environment | ........................................ 21 |
| 3.7 Multi-Hop Reprogrammability | ................................. 21 |
| 3.7.1 Firmware Dissemination | ........................................ 22 |
| 3.7.2 Packet Structure | ........................................ 22 |
| 3.7.3 GATT Characteristics | ........................................ 22 |
| 3.7.4 Utilisation of the Broadcast Mesh | ........................................ 23 |
| 3.8 Developing a Protocol | ........................................ 25 |
| 3.8.1 Limitations of the Protocol | ........................................ 26 |
| 3.9 Bootloader | ........................................ 26 |
| 3.10 Performing the upgrade | ........................................ 27 |
| 3.11 Gateway Node | ........................................ 28 |
| 3.11.1 Web Interface | ........................................ 29 |
| 3.12 Challenges | ........................................ 30 |
| 3.12.1 Flash Memory & Radio Time | ........................................ 30 |
| 3.12.2 Bugs within the Mesh Network | ........................................ 31 |
| 4 Results & Evaluation | ........................................ 32 |
| 4.1 Metrics | ........................................ 32 |
| 4.2 Results | ........................................ 32 |
| 4.2.1 Test Environment | ........................................ 33 |
| 4.2.2 Test Application | ........................................ 33 |
| 4.2.3 Performance over multiple hops | ........................................ 33 |
| 4.3 Space Complexity | ........................................ 34 |
| 4.4 Evaluation | ........................................ 34 |
| 4.4.1 Security & Privacy Concerns | ........................................ 35 |
| 4.4.2 Standardisation | ........................................ 36 |
| 5 Concluding Remarks | ........................................ 37 |
| 5.1 Limitations | ........................................ 37 |
| 5.1.1 Mesh Network | ........................................ 37 |
| 5.1.2 Python API | ........................................ 38 |
| 5.1.3 Protocol | ........................................ 38 |
| 5.2 Future Work | ........................................ 38 |
| 5.2.1 Energy Efficiency | ........................................ 38 |
| 5.2.2 Extensions to the Management Interface | ........................................ 39 |
| 5.2.3 Improvements to the Protocol | ........................................ 39 |
| 5.2.4 Alternative Protocols | ........................................ 39 |
# List of figures

2.1 Share of Internet Connected Devices 
- Page 4

2.2 Architecture
- Page 10

2.3 nRF51 Memory Layout
- Page 11

2.4 BLE Packet
- Page 12

2.5 GAP Service
- Page 12

2.6 GAP Service
- Page 13

3.1 Architecture
- Page 19

3.2 Application Architecture
- Page 20

3.3 BLE Service of Implementation
- Page 23

3.4 Deployment Process
- Page 24

3.5 Mesh Packet Structure
- Page 25

3.6 nRF51 Memory Layout
- Page 27

3.7 Update process in memory
- Page 28
List of tables

2.1 List of Nordic nRF51 Softdevices .......................... 9

3.1 ........................................................................ 26

4.1 Results of Multihop nRF51 updates ......................... 34
4.2 Results of Multihop nRF51 updates ......................... 34

A.1 Table showing the status codes for the custom implemented protocol .................................................. 44
Chapter 1

Introduction

The "Internet of Things" (IoT) is a vision for a seamless integration of technology into the physical environment around us (Gubbi et al., 2013). Sundmaeker et al. (2010) explores the application domains of IoT which range to include medical technology, food traceability, intelligent homes and the independent living of an ageing population. Hodges et al. (2013) advocate the position that the development of more developer friendly tools will allow faster prototyping, testing and deployment of connected devices. This will accelerate the understanding of any remaining challenges in the path to a truly connected world.

There is a clear need for improved developer tooling in order to achieve this goal. Current solutions limit developers to upgrading firmware via either a wired connection or a simple single point to point over the air firmware update. Developers may find this limiting if they are managing a large set of nodes in the field. Additionally, they may not have easy physical access to these nodes. These solutions are not ideal and have been experienced first hand by the author.

The main contribution of this project is to propose a C library for the Nordic nRF51 SoC allowing for updating of firmware over the air. This is the first solution for the nRF51 which allows for multiple hop updates of the application firmware. Furthermore, there is no need for the developer to even be on site at the field deployment of nodes. The introduction of a permanent gateway can allow a developer to remotely manage these nodes. Development of this library included the design of a new protocol. This is then utilised an existing mesh network for dissemination of a new firmware image to the embedded device. Furthermore, the scope of this research
included the development of a python API and web interface allowing interaction with the firmware on these nodes.

There are 4 core sections within this document, each of which is briefly outlined below:

1. **Background**
   In the background we will provide a short introduction to the IoT, analyse existing technologies in the world of the internet of things and provide information on historical technologies which have led to the development of the IoT. Finally, this chapter ends with an introduction to similar work.

2. **Design & Implementation**
   This section begins by outlining a suggested design and architecture for a developer platform build upon an IoT device, followed by an implementation. Details and motivations behind technological decisions made in the process of development are outlined. Finally, a brief overview of the challenges encountered during the implementation process are explained.

3. **Results & Evaluation**
   The results and evaluation review the implementation. In addition to this we will evaluate its performance against the metrics provided from similar projects as well as differing versions of the software, for example to showcase optimisations.

4. **Concluding Remarks**
   The concluding remarks encompass a review, outline the understanding gained, detail limitations, outline how this work has contributed to the development of IoT technologies. Finally suggestions are made for further work.

References and appendices are provided at the end of this document. A list of figures and tables are available prior to the table of contents.
Chapter 2

Background

2.1 The Internet of Things

The phrase the "internet of things" (IoT) was born in early 1999 when it was coined by Kevin Ashton in reference to the introduction of technology in Proctor & Gamble's supply chains (Ashton, 2009). Since then the IoT has grown to mean significantly more, encompassing technology which is becoming prevalent across the office, consumer and industrial worlds. Sundmaeker et al. (2010) explores these application domains including medical technology, food traceability and the independent living of an ageing population.

Gubbi et al. (2013) put forward a strong vision for the Internet of Things where information and communications systems are embedded invisibly all around us. They argue that for this vision to truly emerge, the computing paradigm must move beyond the "traditional mobile computing scenarios". This involves evolving away from the use of portable devices such as smartphones and embedding intelligence into our environment. This can be done by inserting technology into existing everyday items. Gubbi et al. advocate that 3 conditions are demanded from the IoT before it becomes seamless within traditional environments. These are:

1. A complete understanding of the shared context between a user and their appliances

2. Communications infrastructure and software architecture with the ability to in real time process and convey contextual information to where it is relevant

3. Tools for analysis that aim for "autonomous and smart behaviour"
2.1 The Internet of Things

The internet of things is a rapidly growing market. Everyday more and more devices with RFID & BLE connectivity are being built (MacGillivray, Turner, and Lund, 2013). Analysts from the market research firm IDC expect the number of internet connected devices to grow to 212 billion devices by 2020. Vermesan et al. (2011) make predictions regarding the share of internet connected device in each household. These can be seen in figure 2.1. A key trend which can be observed is the steady growth of traditionally unconnected devices gaining connectivity.

Researchers see the internet of things as a natural evolution from the sensor networks. As such the implemented technologies within the IoT are experiencing similar issues to those which have plagued sensor networks (These are outlined on the following page). Want, Schilit, and Laskowski (2013) argue that the future of the internet of things lies with bluetooth low energy, which solves many of these problems. They suggest that the growth of the ecosystem around BLE in the upcoming years will lead to greater adoption of the IoT. Furthermore, Hodges et al. (2013) advocate the position that the development of more developer friendly tools allow faster prototyping, testing and deployment of connected devices. This will
accelerate the overcoming of any remaining challenges in the path to a truly connected world.

The internet of things allows us to the augment the world providing more intelligent and context driven interactions. The sheer size and range of possible applications of the IoT speaks for itself, virtually covering almost every sector of life: medical, consumer, office and industrial (Sundmaeker et al., 2010).

The introduction of connectivity to locations where it has traditionally been unavailable opens up new market opportunities. This new connectivity, in addition to being able to create contextually relevant experiences for the user, allows developers of IoT ‘things’ to deploy to devices where this has previously been an impossibility.

Devices such as toasters, fridges and microwaves may have contained small microcomputers, but the firmware on these devices was fixed at the point of deployment. Utilising this new connectivity allows a developer/manufacturer to quickly and effectively redeploy software. Such functionality provides the consumer both with a better experience and reduces the likelihood of recalls by producer.

Want, Schilit, and Laskowski (2013) and Atzori, Iera, and Morabito (2010) provide us with some insight into the challenges which face the field as a whole.

- Costs of maintaining a large network of sensors
- The difficulty of satisfying the power requirements of the sensor nodes

They suggest that issues which have been barriers to adoption of sensor networks will also be faced by researchers of the IoT. A survey into the state of sensor networks by Yick, Mukherjee, and Ghosal (2008) reaffirms the findings by Atzori and Want - key challenges have yet to be addressed.

### 2.1.1 Uses of IoT Technologies

The domains in which IoT technologies may thrive are numerous. Below we have listed a small subset of the possible users of IoT technology explored by Miorandi et al. (2012):

1. **Smart Homes / Smart Buildings**
   The introduction of IoT technologies into the home can reduce the
2.1 The Internet of Things

consumption of resources of the home by utilising them more intelligently and effectively. Examples of these resources include electricity and water. Furthermore, IoT technologies introduced into buildings can improve the satisfaction level of the humans occupying it. These technologies extend to much more than just the home but also offices, restaurants, etc. Technological enhancements can be introduced into all buildings. The use of sensors with these emerging IoT technologies will provide a key role in the management of deployments within the internet of things. Miorandi et al. (2012) note that there must be a high level of interoperability and standardisation between IoT technologies deployed in these environments in order for them to work most effectively.

2. Health-care

Enhancements to the quality of life can be introduced with IoT for those in existing assisted living situations. Wristbands containing new sensors will be worn by patients. These could provide doctors/nurses with body temperature, blood pressure, breathing activity and multiple other vital statistics. Local medical centres will then receive, aggregate and analyse the information generated from these devices. This will allow for medical staff to build an accurate picture of any patient status allowing them to analyse and respond to conditions as effectively as possible. Furthermore, computer analysis of inputs from patients could trigger warnings and automatically dispatch paramedics to the location of the patient, potentially saving lives.

3. Environmental Monitoring

As the internet of things is a natural evolution from sensor networks which have been traditionally deployed for monitoring, IoT is perfectly suited to providing for environmental monitoring. IoT sensors could be deployed to monitor temperature, wind, rainfall etc. The deployment and utilisation of these technologies is most important in areas where it would be hazardous for humans to operate / manually collect readings. Environmental monitoring technologies will be useful in civil protection scenarios where the access to environmental data in realtime over large scale areas will greatly improve the response and coordination strategies of rescue and repair teams (Miorandi et al., 2012).
2.2 Technologies

2.2.1 Developing for Embedded Systems

For a moment, consider developing an IoT application on embedded technologies. The importance of debugging and development tools cannot be overemphasised. Testing and debugging of software is an inevitable and arduous task, which composes a significant amount of software development time. It accounts for up to 50 percent of the development process (Agrawal, DeMillo, and Spafford, 1990).

While the early stages of development can be performed and tested on a host machine, the final code development must be completed on the target embedded platform. Running a powerful debugger simultaneously as the application can be an unaffordable luxury in embedded systems. It is possible to run remote debuggers from a host machine, but these can often not collect all state detail from the target platform (Akgul et al., 2001). An embedded systems engineer could easily be reduced to debugging using the the LEDs on a device for situations where the debugger interferes with the operation of the device or in cases of the device crashing (ARM mbed Handbook n.d.).

It is important to note the additional complexity of working with embedded systems when considering the development of software for them.

2.2.2 Deploying to embedded devices

There are two methods generally utilised for deployment of firmware to embedded devices.

1. Over-the-wire

The most commonly used firmware upgrade method is the over-the-wire update using for example a USB interface. For this the developer must have direct physical access to the device. Implementations for the flashing of firmware include JLink (Segger and Link, n.d.) and ARM mBed.

2. Over-the-air

Over the air deployments to embedded devices is possible on a number of embedded devices, for example the Nordic Semiconductor nRF51. The current implementation requires a developer to transfer the update from
2.2 Technologies

the iPhone or Android device. This only works over 1 hop, which means the developer’s mobile device must be within approximately 10 metres of the node being deployed to.

2.2.3 Limitations of existing tools

The methods outlined above may work well for very small deployments of IoT devices, but once deployments reach beyond a full tools it becomes limiting to use existing tools. Both require "human-in-the-loop" interactions, which means that for every deployment the deployer needs to be directly involved in the process in order to push an update. Modern development ideals such as continuous integration are not compatible with this existing infrastructure.

2.2.4 Devices

A survey of the bluetooth low energy device market was performed for this project. This was completed to explore which devices, chips and kits were currently available within the marketplace. After reviewing all of the BLE devices, the bluetooth low energy market was found to be composed of two major chipsets:

1. Nordic Semiconductor nRF51822 SoC
2. Texas Instruments TI CC2540

Out of a sample of 27 chips, 7 used the Nordic Semiconductor nRF51, 8 used the TI CC2540, and the remaining were a mixture of either other minor chipsets or the information was unavailable. A full list of the devices and chipsets is available within the appendix.

2.2.5 Nordic Semiconductor nRF51

The Nordic Semiconductor nRF51 is a system on a chip, utilising an ARM Cortex-M0 CPU. The the nRF51 comes with 16kB RAM and 256kB flash memory. The nRF51 supports a standard 2.4Ghz radio, with data rates of up to 2 megabits per second. The radio is designed to allow the use of BLE, ANT, shockburst and many other 2.4Ghz radio standards. The baseband controller also allows for the on-the-fly encryption of payloads using a hardware AES CCM implementation. The architecture is shown in the figure 2.2.
Nordic Semiconductor appreciates the difficulty developers of embedded systems face and provide a number of tools in order to simplify working with their hardware. The largest of these is a set of libraries provided on the device. This is known as a "softdevice". These libraries range from improved interaction with the radio, all the way to writing to flash memory. The clear distinction between essential libraries made by the softdevice affords developers more safety by allowing them to develop applications safe in the knowledge that their application cannot corrupt essential libraries such as the bluetooth stack. This isn't the case on many other embedded platforms.

Nordic have released a number of different softdevices, each support a different a slightly different role. A list of the the current softdevices can be seen in table 2.1.

<table>
<thead>
<tr>
<th>Softdevice</th>
<th>Stability</th>
<th>Protocol Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>S110</td>
<td>Stable</td>
<td>BLE peripheral and broadcaster</td>
</tr>
<tr>
<td>S120</td>
<td>Testing</td>
<td>BLE central</td>
</tr>
<tr>
<td>S130</td>
<td>Experimental</td>
<td>BLE central, peripheral, observer &amp; broadcaster</td>
</tr>
<tr>
<td>S210</td>
<td>Experimental</td>
<td>ANT 8 Channel master &amp; slave</td>
</tr>
<tr>
<td>S310</td>
<td>Experimental</td>
<td>BLE &amp; ANT concurrent operation</td>
</tr>
</tbody>
</table>

Table 2.1 List of Nordic nRF51 Softdevices

**Structure of nRF51 Memory**

The default structure of memory on the nRF51 can be seen in 2.3. The memory layout figure reads from bottom to top, starting with the lowest numbers in memory. The bootloader is 16KB in size on the nRF51, and can be seen at the very top of figure 2.3. It is called when the device is started, and the bootloader configures the device into an initial known state from which it launches the application. Optionally, it can perform advanced pre-start checks in order to confirm that the application has not been corrupted. The bootloader can be modified to perform an arbitrary number of different tasks, but is constrained by the very small allocation of memory that it has. Nordic provide an alternative bootloader that can be used for single hop over the air updates of the firmware on the device.
2.2 Technologies

![Diagram of nRF51 Development Kit / Development Dongle Architecture]

Fig. 2.2 Design of the nRF51 architecture (Nordic Semiconductor nRF51 Documentation n.d.)

2.2.6 Bluetooth Low Energy (BLE)

Bluetooth low energy (BLE), also known as "Bluetooth Smart" is a standard developed by the Bluetooth Special Interest Group (SIG, 2010). It was released under the Bluetooth 4.0 specification in 2010. The S.I.G. observed a number of different trends developing in the computing world and in response it decided to develop BLE. The three major observed trends in traditionally connected devices which led to the development encompass:

1. Wireless technology being deployed to an ever increasing array of devices

2. Rapid increase in the portability of consumer devices in recent years

3. The reduction in cost of wireless technologies
BLE operates on 40 channels, where each channel is spaced at 2Mhz. The channels 37, 38 and 39 are reserved for advertising.

The key change is the substantial reduction in power consumption in comparison to traditional bluetooth. Standard bluetooth has a long handshake process, which can last of up to 100ms. BLE contrasts to this by only needed up to 2ms in order to complete communication with another node (REF).

As outlined by SIG (2010), there are four operating modes of BLE. It is possible for a BLE device to operate in more than one of three modes.

1. **Peripheral**: Advertises and accepts connections, but cannot make connections itself.

2. **Central**: Has the ability to send connection requests to peripheral nodes.

3. **Broadcast**: Advertises itself but cannot accept or make connections.
4. **Observer**: Listens to advertisements but cannot accept or make connections.

All BLE communication is performed via the transfer of small packets. An example packet can be seen in figure 2.4.

![ BLE Advertisement Packet](image)

**Fig. 2.4 Structure of bluetooth low energy packet (Want, Schilit, and Laskowski, 2013)**

Interestingly the design of bluetooth low energy was specifically made so as to reuse as many components of existing bluetooth chips as possible. This had led to the smooth introduction of bluetooth low energy support into many consumer devices with the release of Bluetooth 4.0 (Want, Schilit, and Laskowski, 2013). It is now supported by virtually all modern phones and laptops. This is one of the major benefits of the BLE platform which is likely to have a impact upon the adoption of BLE as an IoT standard.

![ BLE Profile](image)

**Fig. 2.5 Structure of BLE GAP Protocol (SIG, 2010)**
GAP & GATT Characteristics

BLE devices share information using something known as the generic attribute profile (GAP). A visual representation of the structure of a GAP profile can be seen in figure 2.5. Each GAP device advertises services, and each service which is described by a universally unique identifier (UUID). A summary of the services available on a specific peripheral device are included within the advertising broadcast (ADV_IND packet). Constraints on the size of this advertisement packet mean that the list may be truncated if there are many services. A full list of services can be requested from the device by connecting to the GATT layer. A client can request further information on a particular service via the service specific UUID. Services provide information by an attribute described as 'characteristics'. Each characteristic has a value and a set of descriptors, where the descriptors can provide some metadata such as type information. This value is limited to a maximum of 31 bytes. A node in a central role can read values from these characteristics and write to these characteristics if that is enabled (SIG, 2010).

An example of a Heartrate monitor GAP service can is shown in figure 2.6. It exposes two separate services, a heart rate service and a battery service. The heart rate service provides us with a single read only charac-
teristic, which contains the integer value of the bpm (beats per minute) recorded by the device.

2.2.7 Protocols for the IoT

There are a large number of competing protocols within the space of the internet of things, many addressing the varying difficulties of working with an extremely low power device. Regardless of this there is consensus between researchers that the IoT will eventually communicate via an IP based protocol, most likely IPv6 (Jara, Zamora, and Skarmeta, 2012).

This can be seen by the work from the IP working group, concur with this view. They have implemented a person area network based upon IP for standard bluetooth 4.0, and have begun drafting the same for BLE. There is a current IETF working group led by Nieminen et al. (n.d.) developing a bluetooth specification for bringing what is known as IPv6 to BLE.

Unfortunately current implementations of IPv6 are not particularly well suited to low power devices, as they still have a high overhead. Some embedded devices do not even have the power or capacity to operate a basic IP stack, and there is some trouble using 64 bit IPv6 addresses on the 8 and 16 bit TI CC2450s. There are some efforts to provide easy IPv6 support across a set of devices in implementations such RiotOS by (P. Levis et al., 2005).

Trickle Dissemination Algorithm

The RFC 6206 "Trickle Algorithm", developed by Philip Levis, Clausen, et al. (2011) has been utilised in a number of IoT scenarios. Levis argues that it has a number of benefits which make it highly suited for the use on lossy low power networks. The algorithm takes into account how often information changes and the density of the nodes around the current node to decide on how often a particular node should rebroadcast information. Trickle operates over time intervals, but does not require any synchronisation between nodes (Philip Levis, Brewer, et al., 2008). The trickle algorithm is designed to lead to eventual consistency across the entire network, quite similar to how the "Gossip" protocol works.

This is a highly efficient algorithm for this particular kind of network and has already been utilised with success in a real world deployment. This was observed in the ‘Trio’ management network by Dutta et al., 2006.
very useful for sensor networks as it makes no topological assumptions and imposes an extremely low overhead when the network is consistent/stable.

2.3 State of the Art

2.3.1 TinyOS
TinyOS was developed in UC Berkeley, and is one of the first sensornet OSes (P. Levis et al., 2005). TinyOS provides a flexible architecture coupled with a low resource consumption. This is possible due to the modular nature of 'components' utilised by TinyOS. These are wired together to create an application at design time. The execution model of TinyOS focuses around tasks and interrupts. TinyOS provides 'service distribution' which the provision of groups of components together as services, which improves system reliability. Unfortunately there is a severe limitation of TinyOS :- the compiled image is monolithic. While this is done for efficiency, it means that no single individual component can be separated for reprogramming.

2.3.2 Contiki
Contiki is developed by the Swedish Institute of Computer Science in order to address what it sees as the key weakness in TinyOS (Dunkels, Grönvall, and Voigt, 2004). This is done via the introduction of dynamically loadable modules. The use of dynamically loadable modules makes Contiki interesting for context of reprogrammability. Contiki supports cooperative multithreading.

2.3.3 Similar Work
A review of literature was performed to find relevant and similar work, some of which is outlined below:

"Trio" Wireless Sensor Network
The most similar work built to date is the "Trio" wireless sensor network implemented in 2005 by Dutta et al. (2006). It was an outdoor network of nodes powered by photovoltaic cells. The Trio network was built up of 557 nodes, seven gateway nodes and a single root server. It used the trickle algorithm for the propagation of data within its mesh network. While
the development of a management tool for nodes and deployments was undertaken, it was not the original goal of Dutta's research, it provides many insights. These include details such as weaknesses in deployment and scalability of the network. One of the shared aims between this proposal and "Trio" is to reduce the number of human-in-the-loop operations which are performed during the regular operation of the network. "Trio" predated the development of Bluetooth Smart and was implemented utilising 802.11, traditionally known as WiFi.

The three gateway nodes connected to a root server which could view the network as a whole and allow active querying or passive monitoring of the network.

While useful unfortunately the paper published on "Trio" did not provide statistics regarding the success rates of deployments and updates.

**ExScal: Elements of an Extreme Scale Wireless Sensor Network**

Research by Arora et al. (2005) incorporated the development of 1000+ node sensor network. This network was funded by DARPA built in order to detect any vehicles entering the area covered by the mesh network. Such a network introduces a number of unique challenges in terms of scalability and management. A core software 'Nucleus' was develop to provide basic power management and deployment capability. They introduced two distinct managerial nodes within the network. These are outlined below:

1. **System Operator**  
   Which can via a gateway communicate into the network and perform operations

2. **Local Manager**  
   Self selected node that detects and corrects low level faults which occur in nodes

A deployment failure rate of 5.37% was observed across the network during the live operation of the network, in addition to this a similar 'reprogramming' fault was observed at an occurrence rate of 5.5%.

**"MoteLab": A Wireless Sensor Network Testbed**

Werner-Allen, Swieskowski, and Welsh (2005) developed "MoteLab" in Harvard in order to develop, deploy, and debug applications on realistic
large-scale sensor networks. Werner-Allen argues that these are logistical challenges which face many researchers entering the field of wireless sensor networks.
Chapter 3

Design & Implementation

3.1 Requirements

This system, as envisioned, has a number of key requirements. These will provide us with a much clearer platform from which to evaluate our design:

- **Version Control**: A central version control database manages various versions of the firmware images for deployment.

- **Scope selection**: Allowing administration to select one or a group of nodes for reprogramming.

- **Code dissemination**: The protocol transmits updates from a source node to a targeted node.

- **Code validation**: The new program should be received on the target node without any errors.

- **Automated discovery**: Nodes should be able to join the network without any reprogramming or manual intervention to the network.

- **Management Interface**: This is an interface provided to the network operator in order to perform reprogramming tasks across the network.

3.2 Developing for the IoT

Developing applications for the internet of things requires significant knowledge of the embedded devices being used. Furthermore, as outlined the normal activities of a software developer can be significantly
limited, eg. debugging tools. Working with such limited devices introduces a number of challenges. Some of these challenges were encountered by the author and are outlined in the "Challenges" in section 3.12.

### 3.3 Architecture of IoT

D. Bandyopadhyay and Sen (2011) outline the layered architecture of the IoT, which can be seen in figure 3.1. The figure shows us the field data acquisition at the bottom, to the application layer at the top. This layered architecture has a distinct division where the lower two layers are concerned with data acquisition and the upper two layers are concerned with the utilisation of this data.

![Layered architecture of the Internet of Things](image)

**Fig. 3.1 Layered architecture of the Internet of Things**

### 3.4 Structure of Solution

In order to fulfil the aims of this project, which encompass a fully featured remote reprogramming solution, solutions must be developed for every single layer in the internet of things. The design of these layers is outlined below:

#### 3.4.1 Edge Technology

Edge technology is composed of physical hardware which has been deployed in the field. It is required for information collection, communica-
3.4 Structure of Solution

tion, control & actuation. This includes sensors, IoT devices and gateway nodes. In figure 3.2 we can see a more comprehensive view of components of such a node deployed in the physical environment. The proposed implementation utilises the radio of the device combined with a protocol such as bluetooth low energy.

Fig. 3.2 Structure of an "Internet of Things" application (S. Bandyopadhyay et al., 2011).

3.4.2 Access Gateway

The access gateway exists in order to allow external management software to interact with known nodes deployed into the BLE network. This abstracts away working with the internal network protocol, and converts the communication into IP. Traditionally this would be implemented as a computer which would be connected to the internet on one interface and have a subnet of internet of things nodes on the other interface. D. Bandyopadhyay and Sen (2011) makes the argument that protocol gateways are inherently "complex to design, manage, and deploy" and that in the future with end to end IP, there would be no protocol translation gateways involved.
3.4.3 Application Layer

The application layer is the management interface, which is built to expose the functionality of the network to the system administrator. It lies at the top of the stack and is responsible for communicating with access gateways. For the implementation of the tool suggested by this project, the application layer will involve the preparation of application for delivery to nodes in the network.

3.5 Hardware

A review of hardware devices was performed as part of this research. An outline of the research is available in section 2.2.4. It was decided to use the Nordic Semiconductor nRF51 system on a chip for this implementation. The School of Computer Science & Statistics of Trinity College Dublin kindly provided a set of sample Nordic Semiconductor nRF51, including both the USB dongle and the Arduino compatible development kit. As outlined in section 2.2.4, these are highly capable devices which implement BLE. An outline of the architecture can be reviewed in section 2.2.5.

3.6 Language & Environment

The nRF51 supports both the development of applications through C and C++. It was decided to develop in C due to a number of useful libraries being available in the C language. This C code is then compiled and assembled in a 'hex' file which can be written to the flash memory of the nRF51. These hex files are being utilised by the reprogramming software.

3.7 Multi-Hop Reprogrammability

Multiple hop reprogrammability of nodes is a feature currently lacking on a number of recently introduced IoT platforms, including the nRF51. Multiple-hop reprogrammability means that nodes within an IoT deployment can be remotely be modified to run different firmware.

A review of the available technologies for the nRF51 exposed a mesh network built by researchers from the Norwegian University of Science & Technology. It was developed in order to provide broadcast mesh network
3.7 Multi-Hop Reprogrammability

functionality to the nRF51. This implementation utilises a quite low level implementation in order to function correctly with the S110 softdevice. This approach was used over alternatives for two major reasons:

- The S110 softdevice required by the nRF51 broadcast mesh is stable and the S130 softdevice required for more involved approaches is only of experimental stability.

- The use of mesh networks has existing proven success with use by management software Dutta et al., 2006 for 'Trio' and Arora et al., 2005 for 'ExScal' in real world scenarios.

3.7.1 Firmware Dissemination

There were a number of benefits leading from the utilisation of the broadcast mesh network for the dissemination of firmware for this implementation. When building a multi-hop management platform there is a need to introduce some form of transport and routing across a set of nodes.

3.7.2 Packet Structure

The packets used by the mesh network deviate slightly from the standard Bluetooth specification. A diagram of the packet structure can be seen in 3.5. There is a slight modification in that the advertisement address within the Bluetooth packet is the advertisement address of the node that originally added the packet to the network and not the node that has recently rebroadcast the packet.

The structure of these low energy Bluetooth packets leaves us with 30 bytes of space to utilise for the implementation of a protocol.

3.7.3 GATT Characteristics

The broadcast mesh allows a developer to add up to 155 characteristics. These characteristics are synced directly into the softdevice GATT characteristics, exposing them for access as a GAP service. This allows devices connected to the nRF51 using the standard Bluetooth low energy protocol to directly read and modify the characteristics within the mesh network. These characteristics are then broadcast to the mesh network via the radio using the trickle algorithm. One of the larger broadcast mesh limitations is that the mesh network operates with a severe performance penalty if it
utilised more than 'several' key value pairs. This was due to the additional strain on the devices / network to transport the additional key value pairs. Due to this an informed decision was made to limit the number of characteristics to 14. This ensures that only a limited number of key value pairs would be updated at any one point in time.

3.7.4 Utilisation of the Broadcast Mesh

The infrastructure which the mesh library provides on the Nordic nRF51 is limited to two components:

1. A set of functions for retrieving and interacting with the mesh data
2. An event handler which is called when a value is updated in the mesh network

The event handler was where the majority of the heavy lifting implementation of this management software was introduced.

There are 14 characteristics available for utilisation for communication. These are designed to be used for communication between a specific node in the network and the deployed gateway node. For the clarity this paper will call these GATT characteristics 'channels' from this point onwards. One of these channels is reserved for simplex broadcast communication
between the application and all nodes within the network. Nodes are implemented as such to never modify the value in this 'zero channel'. Examples of commands within this channel include directing specific nodes to start communicating with the server over a defined channel or perform a number of varying operations upon groups of nodes within the mesh network. The remaining 13 channels are used for half-duplex communication between nodes in the network and the application server.

Each node in the mesh will receive an event call for every update to a characteristic within the mesh. For each incoming message a bitmask is used to extract the target address. The device's current address is then checked against this value, and if it matches the rest of the contents of the packet is checked whether it is at the relevant point to perform some actions within that specific state machine.

Furthermore, the scope of this project included allowing external BLE devices can access a GAP service where the GATT characteristics from the mesh network can be viewed and modified. This was key to the development of the platform as it allowed other external devices to manipulate the values which were being broadcast from one specific node in the network.

As the application server does not have any knowledge about the packet travel time between the access gateway and the targeted node, it expects an acknowledgement for each individual packet of data which it transfers to the node in question.
3.8 Developing a Protocol

Using the 30 bytes that we have available, this paper sets out a design for a protocol utilising the nRF51 mesh network the central gateway node and the peripheral nRF51 nodes to communicate. Using channel structure outlined in section 3.7.4, this protocol is used for communication with nodes in the network. The first 5 bits of the communication are reserved for the operation code. The operation codes for the server to node communication are listed in Appendix A.1. Following the operation code are 3 bits reserved for undefined future use. These were left for use by optimisations, and improved scope selection. For example this could be used to indicate that the target address is the ID of a group of nodes rather than just an individual node. Following these 3 bits are two bytes containing the target address of the current operation.

The target address is composed of the first 4 bytes of the mac address of the device. This was the addressing scheme used by the mesh network when advertising. Reusing this structure simplified the debugging of the network. Conflicts of addresses can be handled by attempting to ping a device’s own address.

These channels are used for half duplex communication.

As outlined within the background, there is a strong consensus between researchers that IPv6 will eventually become a standard between internet of things devices. While we share this view, due to the limitations put...
upon this project by the broadcast mesh network a custom protocol was implemented.

### 3.8.1 Limitations of the Protocol

Some limitations are imposed by the method of addressing being used by the mesh network. Each device advertises itself by the integer value of the first 4 hexadecimal digits describing the device. Theoretically this allows for the creation of a mesh network with a maximum of 65535 devices, which seems more than sufficient for the networks that we would be building. An improved implementation of the protocol would mean introducing support for the nodes to scan the network and change their address based upon the results of this scan.

The library implementation does not include any timers, and hence any communication with a server will never timeout. This can lead to the nodes entering an unknown state after lossy or unsuccessful communication.

### 3.9 Bootloader

The implementation of this project required the development of a custom bootloader. This custom bootloader is a modified version of the nRF51 DFU BLE bootloader. In order to suit our purposes the bootloader's BLE and DFU capability is removed in its entirety. The bootloader is modified to begin the upgrade process when the app resets into the bootloader app upgrade mode. The operation of the bootloader is discussed in section 3.10.
3.10 Performing the upgrade

The upgrade process used is similar to the device firmware upgrade process which nRF51 uses in its default single hop over the air transfer of information.

The transaction works as follows:

1. A connection is confirmed between the gateway and the node in question, using a mesh network.

2. Some basic information required for the update, such as the expected size of the update and the CRC (Cyclic Redundancy Check) of the image are transferred to the target node.

3. The image is then transferred across the network. This is performed in chunks, and each chunk is acknowledged individually by the targeted node. Within this implementation each chunk is composed of 16 bytes.

4. Finally after the image has been fully transferred, an activation packet is sent from to the target node.
5. The targeted node verifies the image using the CRC information received in earlier communication. If there is any discrepancy between the CRC generated and the CRC provided the update is aborted and the entire transfer is discarded.

6. If the CRC check completes successfully the device resets into the custom bootloader.

7. The bootloader starts up and checks whether it has been started in normal operation or into upgrade mode. If upgrade mode has been activated by the application, the bootloader begins the application swap.

8. The application swap operates by swapping the contents of Bank 1 into Bank 0. This can be seen in figure 3.6.

9. On completion of the swap the bootloader verifies and start the application in Bank 0.

### 3.11 Gateway Node

There is only one tier of nodes implemented within this mesh, and commands are issued into the mesh network via GAP services. Protocol commands can be issued from any BLE compatible device.
A python application running upon the Ubuntu 14.04 OS platform was used for adopting the central node role and testing this implementation. This python is designed to be as modular as possible, which means that any python application could be extended to deploy firmware into a deployed network.

This was implemented on the Ubuntu 14.04 platform utilising the BlueZ gatttool terminal tool. All interactions with this tool are completed via command line calls out from the python script.

This implementation does introduce some limitations such as the fact the only one update can be run at a time from the main node. This is due to the fact that during an update there is a connection made between the Ubuntu BLE device and one specific node within the network. The nRF51 device can only sustain one such connection at a time. Any interactions with the API do not need to have any knowledge of the network with the exception of the MAC address of a nearby node, and the addresses of any nodes which an operator may want to perform operations upon.

### 3.11.1 Web Interface

The development of this project also incorporated the development of a web interface for the administration of the nodes within the network.

The features of the prototype management platform include the abilities:

- Authentication of users
- Ping any known node within the mesh network
- Upload a hex application file and deploy it to any known node of your choice
- Check the latest known state

This web interface was implemented with a number of different technologies. The python web micro-framework known as "Flask" was used for this purpose. A SQLite database was used to store both the user credentials and the details regarding known nodes. Database records for each node store information on: (a) the latest version of firmware known to exist on each node and additionally (b) a unix timestamp detailing the last time
which an interaction with the device were recorded. This visual web interface interacts with gateway python API in order to communicate with the nodes.

One feature of particular interest in the web interface is the fact that after the upload of firmware, the user is presented with a webpage which resembles a terminal console. This terminal console window shows the user an accurate progress report of the image transfer. It details the transfer of an image to the packet by packet level.

### 3.12 Challenges

Throughout the development of this implementation the author experienced some challenges

#### 3.12.1 Flash Memory & Radio Time

This was the largest challenges encountered during the development of this implementation. Almost one week of developer time was spent attempting to debug this issue. The architecture and implementation of the S110 soft device introduce a number of restrictions upon the simultaneous operation of the radio, CPU and flash memory.

Attempts were made to write the packets incoming from the radio to flash memory. This failed with an inconclusive error message. Attempts were made to solve the issue by running the debugger. Unfortunately the debugger interfered with the operation of the mesh network, which would render the process void.

At the point this issue occurred, packets containing the image were being arriving successfully on the node.

Once it became clear that the failed writes to memory were due to radio interference, attempts were made to reduce the radio time the mesh network utilised. This alleviated the problem to a certain extent, and some packets were written to flash memory successfully. Unfortunately the majority of packets still failed to write.

This issue was finally fixed by the introduction of a limited number of retries writing to flash. Effectively we keep retrying writing to flash until we call when the radio is not in operation.
3.12 Challenges

3.12.2 Bugs within the Mesh Network

A number of different bugs were discovered during the implementation process. The largest bug was the mysterious occasional failing of all communication on the mesh network after about 8 minutes. Investigation uncovered that the if the mesh was being heavily utilised the global network time overflows unsafely.

This bug meant that the mesh stopped functioning after 7.1 minutes when the 8 bit integer overflowed.

A temporary fix was implemented where the global time counter on each of the nRF51 devices was modified to use a 64 bit integer. This temporary fix, while effective is not necessarily the most efficient fix possible upon the network. The bug was reported to the repository maintainer and will hopefully be resolved within the near future.

Conflicting Packets

On occasion a conflict may occur between a set of packets. During the regular operation of the mesh, any update to the value of a particular key on the mesh will be propagated by (1) incrementing the version number by one and then (2) broadcasting this value. If two nodes separated by multiple hops in the network do this simultaneously to the same value this causes what is known as a conflict. This will reach the event handlers of the nodes which are a few hops the nodes from which these updates originally originated from. The implementation of the mesh network does not provide any conflict resolution code, but instead will throw an event known as an "RBC_MESH_EVENT_TYPE_CONFLICTING_VAL" event. A decision was made within the project not to attempt to handle these conflicts by nodes within the mesh, as there is no suitable conflict resolution heuristic. Furthermore, the assumption was made that this kind of event is unlikely to occur as within this implementation any communication 'channel' used only be used for half-duplex communication. This means that at least one node is always awaiting the result of the other, rather than utilising the same channel simultaneously.
Chapter 4

Results & Evaluation

4.1 Metrics

Wang, Zhu, and Cheng (2006) outlines three key elements of an IoT reprogramming service. These are outlined below:

• **Time & Space Complexity**
  Due to the limited nature of the hardware being utilised for devices in the IoT, the time and space complexity of the solution needs to be well suited for these devices. For example the nRF51 provides 256kB of flash memory, but only approximately 128kB of it is available for applications.

• **Delivery**
  A successful reprogramming requires the entirety of the flash image to be delivered to targeted nodes eventually. This can be limited due to failures in transmission of packets, node failures and signal collisions.

• **Energy Efficiency**
  The service should be energy efficient. This is important on devices where there is limited battery power available and energy cannot be harvested from the local environment.

4.2 Results

This project has reviewed 2 of these 3 metrics of an IoT reprogramming service: "Delivery” and "Time & Space Complexity”. Unfortunately the
4.2 Results

The analysis of the energy efficiency of the chip requires special equipment which was unavailable to the author.

4.2.1 Test Environment

All of the tests outlined within this chapter were undertaken in the suite of rooms and laboratories of the School of Computer Science and Statistics at Trinity College Dublin.

4.2.2 Test Application

An example application was built, which is a very simple "Hello World" application integrated with the library of the reprogramming implementation. It will momentarily flash LED1 on startup, and then the application will idle in its event handler. Unless otherwise specified, this is the application used within the tests.

4.2.3 Performance over multiple hops

Purpose of Test

One of the key new features introduced by the solution outlined within this paper is the introduction of multiple hop updates. In order to evaluate the performance of the multiple hop we compare the following:

- Nordic nRF51 firmware update over USB
- Nordic nRF51 firmware update over the Air utilising the official iOS application.
- This implementation using over one hop.
- This implementation over an increasing number of hops.

Ensuring Node Hop Count

When collecting statistics for the performance of the update process over multiple hops, it is important to ensure that the packets do not have a shorter path due to the structure of the nodes in the mesh network. The hop count was confirmed by placing the node at a location, removing the intermediate node and ensuring the new node was out of range. The
removed node was then replaced and it was confirmed that the new node was responding to discovery messages.

**Results**

<table>
<thead>
<tr>
<th>Test</th>
<th>Hops</th>
<th>Average Transfer Time (Min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRF51 USB</td>
<td>N/A</td>
<td>0.1 min</td>
</tr>
<tr>
<td>nRF51 OTA</td>
<td>N/A</td>
<td>1 min</td>
</tr>
<tr>
<td>Implementation</td>
<td>1</td>
<td>16:32</td>
</tr>
<tr>
<td>Implementation</td>
<td>2</td>
<td>18:15:81</td>
</tr>
<tr>
<td>Implementation</td>
<td>3</td>
<td>18:00.2</td>
</tr>
</tbody>
</table>

Table 4.1 Results of Multihop nRF51 updates

A tabulation of the results of the the multiple hop upgrades can be seen in table 4.2.

### 4.3 Space Complexity

<table>
<thead>
<tr>
<th>Test Application</th>
<th>Compiler Optimisation</th>
<th>Image Size</th>
<th>Transfer Time (Min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World app</td>
<td>O0</td>
<td>66kB</td>
<td>17min</td>
</tr>
<tr>
<td>Hello World app</td>
<td>O1</td>
<td>50 kB</td>
<td>12min</td>
</tr>
<tr>
<td>Hello World app</td>
<td>O2</td>
<td>47kB</td>
<td>11.5min</td>
</tr>
<tr>
<td>Hello World app</td>
<td>O3</td>
<td>47kB</td>
<td>11.5min</td>
</tr>
</tbody>
</table>

Table 4.2 Results of Multihop nRF51 updates

The space complexity of the library is important on devices that operate with such a limited computing power. The space complexity will also vary depending on the optimisation level of the compiler used. This is an important factor as operators of the network will want to debug software on these nodes, which normally requires compiling with all optimisations turned off. Optimisations can interfere with the normal operation of debuggers.

### 4.4 Evaluation

Several metrics are generally used in order to evaluate a remote reprogramming service such as the one outlined within this paper. Among these are reliability, coverage and autonomy. Together these ensure the correctness of a reprogramming service.
4.4 Evaluation

- **Reliability**
  The entire contents of the update should be received without error on the target node.

- **Coverage**
  If the targeted node is within range it must eventually be reprogrammed.

- **Autonomy**
  The reprogramming should be completed minimal interaction of a human operator. Preferability the operator should remain at the base station.

These key elements of correctness allow us to define a number of metrics which are important to the efficiency of a program.

### 4.4.1 Security & Privacy Concerns

Both security and privacy of communication across any IoT network is of major concern. Gubbi et al. (2013) makes a clear argument that this will be of particular concern when networks reach large scales, where there may be significant malicious actors or adversaries. Encryption ensures data confidentiality against outside attackers, whereas message authentication codes (e.g. CRC) ensure data integrity and authenticity. They note the importance of developing protocols which would lay the foundation for more advanced cryptographic add-ons to be developed. Deluge is one example of a cryptographic protocol add-on currently in development.

While this project notes the concern for security and privacy as an active research area within the IoT, this project will not focus upon these. The implementation focused solely on building a platform with a view that encryption can be added as an additional layer at a later stage. This does, however, introduce the requirement that the communications layer is to be structured in a modular fashion. The implementation of a cloud application introduces the aspect where security and privacy concerns need to be respected. Lori (2009) notes that the user of cloud technologies means that device users are transferring increasing amounts of data into the cloud. Users will no longer know where their data is stored, and with whom they are sharing this data storage. Furthermore, there are concerns as to the type of encryption schema and access policies being used by the hosting provider.
There is still much work to be done to clarify the whether a new legal environment is needed for this entirely new field of IoT devices and applications (Weber, 2009). Weber et al. makes it clear that there is a need for regulatory and technological improvements in the coming years.

### 4.4.2 Standardisation

Coetzee and Eksteen (2011) argue that in order for the internet of things to succeed, standards need to be introduced to allow maximum interoperability. They outline how the building of standards allows the world of IoT to start sharing data and utilise this to build more intelligent contexts. Miorandi et al. (2012) furthers this opinion, by acknowledging that the "semantic interoperability" is one of the key system-level features which an internet of things device needs to support. Various sensor information could be useful when shared in these scenarios, especially from sensors where a single input signal would not provide a sufficient actionable piece of information, but the conjoined data from a number of different sources would together indicate some issues.
Chapter 5

Concluding Remarks

A successful implementation of a proof of concept was developed, showing that the implementation of a deployment management tool for the nRF51 is possible. At its core it provides a platform from which developers can push updates, and in addition to this developers can utilise the network for standard communication between nodes, network analysis and much more.

While a number of weaknesses exist, the implementation shows promise for embedded systems engineers. Tools such as this one will allow them to more easily adopt modern development methodology such as continuous integration.

5.1 Limitations

5.1.1 Mesh Network

The key limitation is the fact that mesh requires the radio to be on all of the time. This consumes large amounts of energy which is not particularly feasible in any bluetooth low energy network, unless these nodes have access to a persistent power source.

There were some major difficulties faced during the implementation of the project. These were largely focused around the limitations of the nRF51 hardware. As outlined in the architecture section of this document the simultaneous use of the radio and the CPU is not possible.

The original implementation of the mesh network did not utilise a nRF51 soft device. The integration of the soft device was necessary for the implementation outlined within this paper. This proved to be a challenge
due to the how the mesh network handled events. The mesh library provided its own event IRQ handler. The nRF51 platform only supports one such IRQ handler. A patch had to be made to the soft device IRQ handler in order to ensure continued operation of the mesh network.

5.1.2 Python API

The python API is tied to the Ubuntu's BlueZ 'gatttool'. This is limiting, as it means that the python API must be run on the Ubuntu platform. Furthermore, if the administration panel is only configured to use the specific mac address of one node in the network. This means that the panel can only perform one action at a time. Any further actions will time out while the first action is run.

5.1.3 Protocol

There are limitations to the protocol design for this implementation of the multi-hop reprogrammability of nodes in the network. The current implementation of protocol disseminates every update to every node within the network. This is not efficient and will lead to faster depletion of power on all nodes.

5.2 Future Work

The evaluation of the results of the described implementation shows promise for further development. Outlined below are a number of aspects of the project that deserve further attention.

5.2.1 Energy Efficiency

The energy efficiency of a given protocol is important for comparison with other tools in the field. Due to the unavailability of debugging hardware for the checking the energy consumption of a node, this paper does not have power metrics available. This should certainly be investigated in the future in order to allow for a comparison to other reprogramming tools.
5.2 Future Work

5.2.2 Extensions to the Management Interface

Extensions to the management interface of the software could provide features to ease the development process for embedded system developers. One example would be the integration of a git based deployment architecture, providing a deployment platform as a service. This would mean that with each git push to a special git remote, a remote server would build the firmware image. This firmware image would then be automatically deployed to a specified set of nodes.

A further extension to the existing management API would be deploying tests to a network of nodes. This could be provided as an automatic service to the developer, in the form of unit tests being run. This would allow the operator of the network to develop using the principle of continuous integration.

5.2.3 Improvements to the Protocol

One of the major limitations outlined within the evaluation notes that the protocol involves every node within the network when performing any action on the network. While this is not an issue if every node in the network was targeted for simultaneous dissemination of a protocol image, it becomes an issue if the network operator decides to select a more specific scope. For example, the operator may decide to reprogram all nodes in the network of a specific type.

5.2.4 Alternative Protocols

A key issue to the existing implementation is that a broadcast network is not necessarily the best possible implementation of technology for the purposes proposed by this project. A major concern would be the modification of this implementation into something more suitable for a point to point update. A number of different alternative implementations can be created. One such implementation could be building a single hop over the air deployment of firmware between nodes. Nodes could connect to nearby nodes as central BLE nodes and share recently deployed firmware. This would require the implementation of a routing algorithm in order to be successful.
5.2 Future Work

5.2.5 Support of Additional Platforms

The benefit of this the implementation outline depends greatly on large clusters of these interoperable devices being deployed. The development of compatible partner versions on similar platform, perhaps for the Texas Instruments CC2450. The TI chip is being used by 30 percent of BLE devices under a survey conducted for this paper. An additional implementation for this device would encouraged by the author of this paper.

5.2.6 Modular Reprogrammability

Implementation of differential or modular reprogrammability would be very useful addition to the implementation. The current implementation requires the monolithic deployment of an entire application image.
References


Hodges, Steve et al. (2013). “Prototyping Connected Devices for the Internet of Things”. In: *Computer* 46.2, pp. 26–34. ISSN: 0018-9162. DOI: 10.1109/MC.2012.394.


Sundmaeker, Harald et al. (2010). “Vision and challenges for realising the Internet of Things”. In: ResearchGate. DOI: 10.2759/26217.


## Appendix A

### Mesh Protocol Status Codes

<table>
<thead>
<tr>
<th>Protocol Command</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESH_NOP</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MESH_CONNECTION_REQUEST</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>MESH_CONNECTION_REQUEST_ACK</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MESH_DISCONNECT_SERVER</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MESH_DISCONNECT_CLIENT</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>MESH_REQUEST_STATUS</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>MESH_REQUEST_STATUS_ACK</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>MESH_START_IMAGE_TRANSFER</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>MESH_START_IMAGE_TRANSFER_ACK</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>MESH_DATA_IMAGE_PACKET</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>MESH_DATA_IMAGE_PACKET_ACK</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>MESH_DATA_IMAGE_REQUEST</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>MESH_IMAGE_TRANSFER_SUCCESS</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MESH_IMAGE_TRANSFER_SUCCESS_ACK</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>MESH_IMAGE_ACTIVATE</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>MESH_IMAGE_ACTIVATE_ACK</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>MESH_CLIENT_ERROR</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1 Table showing the status codes for the custom implemented protocol