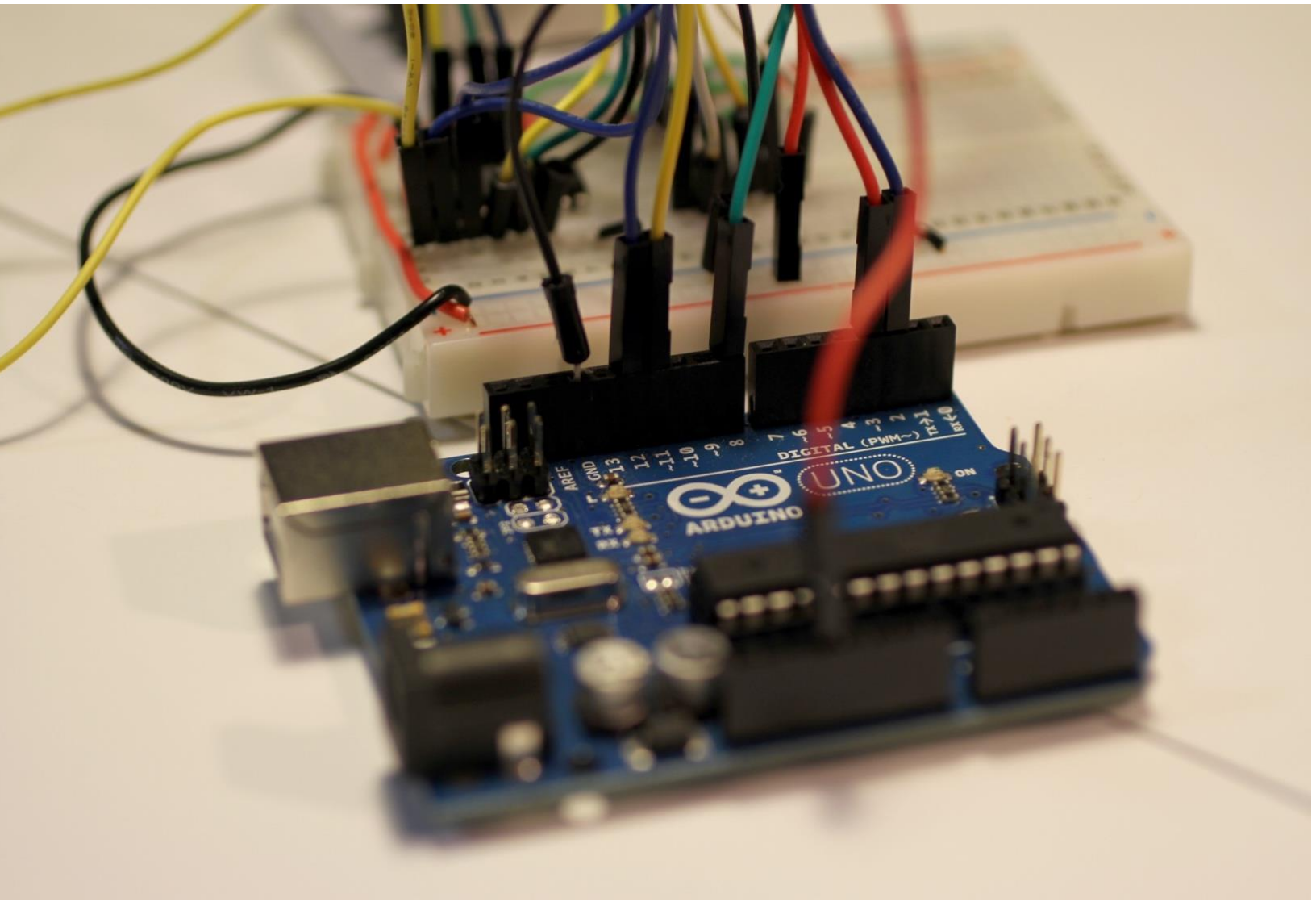
Energy Management in Low Power Wireless Sensor Networks

Final Design

Cloud Seven Consulting

THE UNIVERSITY OF WESTERN AUSTRALIA



Energy Management in Low Power Wireless Sensor Networks
Team 14

Authors
Peter Bouvy (21299044)
Yung Ren Chin (21247413)
Aaron Hurst (21325887)
Khanh Tan (Jamie) Phan (21326604)
Matthew Ramanah (21317297)
Jake Sacino (21132001)
Andy Ta (21317377)

Final Design

Project Partner:
Mr Mark Callaghan, ATAMO

Supervisor:
Mr Marcus Pham

Unit Coordinator:
Dr Sally Male

Group Meeting Day and Time:
Thursday 4pm

Word Count:
????

Word Limit:
24,500

The team agrees to share the presentation time and all members agree to receive the same presentation mark if the opportunity to demonstrate capabilities is deemed uneven by the assessors.

Version 5.1

[1]*Cover page image kindly provided by Visual Hunt [1]*

Revision History

Date	Version[2]	Description	Author
29/09/2017	1.0	Creation of initial template	Jake Sacino
30/09/2017	1.1	Added Introduction; Added preface to Requirements section	Jake Sacino
04/10/2017	1.2	Added Sensor-Host introduction	Jamie Phan
21/10/2017	1.3	Stakeholder Engagement	Aaron Hurst
22/10/2017	1.4	Front-End portion of Abstract	Andy Ta
22/10/2017	1.5	Finish Abstract, Design Philosophy, Join Implementation	Matthew Ramanah
23/10/2017	1.6	Updated Network Manager Sections	Peter Bouvy
23/10/2017	2.0	Added 5.2.6: Sample – Packet Payload Formation	Aaron Hurst
24/10/2017	2.1	Updated Hardware Design Sections & System Cost	Yung Chin
25/10/2017	2.2	Updated Cloud Database Sections & Design Outputs	Andy Ta
26/10/2017	2.3	Added Eterna Flash section	Jamie Phan
26/10/2017	3.0	Added Interface Design section	Jake Sacino
26/10/2017	3.1	Added Power Supply Design	Jamie Phan
26/10/2017	3.2	Added Crystal Configuration	Jamie Phan
27/10/2017	4.2	Added Risks & Contingencies section; updated Structure & Contributions	Jake Sacino
27/10/2017	4.3	Updated Data Acquisition, Sensor Configuration and Mote Configuration sections	Aaron Hurst
27/10/2017	5.0	Added Testing section	Aaron Hurst
27/10/2017	5.1	Report editing	Aaron Hurst

[2]*Incrementing the version by 0.1 denotes a minor change; incrementing by 1.0 denotes a significant change*

Abstract

Wireless sensor networks (WSN) are often used to monitor a wide range of physical and environmental conditions such as temperature, pressure or flow [2]. A WSN system consists of several spatially distributed sensors, each forming a single node in the network. Nodes are able to communicate with each other through a complex mesh to a gateway, providing bi-directional wireless connectivity over a short distance [3].

Power management is one of the major challenges facing wireless sensor networks [4]. Cloud Seven Consulting (CSC) have integrated a third party WSN with an ATAMO Arduino-based development platform, periodically reporting sensor readings to a cloud database which is accessible through a graphical user interface. Whilst functional communication between each component of the project is the main driver of the design, further emphasis has been placed on the WSN power management system to prolong the battery life of the sensors.

CSC utilised a duinoPRO board to drive the sensors and frame the payload appropriately, minimising the sensor battery drain. A Dusty module was then used to send the payload to the mesh network. A duinoPRO, Dusty module and sensor together form a single node in the network, also known as a mote. Communication within the mote was achieved using the universal asynchronous receiver-transmitter (UART) protocol.

Communication between motes, the gateway and the database was achieved by leveraging the SmartMesh Internet Protocol (IP) [5]. Dynamic power management was implemented through a complex scheduling routine in parallel with the regular operation of the mote, further enabling CSC to maintain tight constraints on battery energy usage. All routines within the mote were designed using standard Arduino libraries and available duinoPRO libraries in accordance with the relevant application programming interface (API).

CSC integrated a cloud-database in conjunction with a web application to visualise and store data from various motes. Utilising Amazon Web Services, Django and Python, CSC has developed a system with end-to-end functionality for proof of concept. Facilitating for remote monitoring, the premise of the cloud-database is to store data on a multi-user accessible platform. Creating a medium for remote data analytics and cost-effective data sharing methods.

In piping the data from the network manager to the cloud-database, it has created opportunity for CSC to deliver a cloud integrated web application. With a deep consideration for time-series data, CSC has developed a user-interface which displays multiple motes in a simplistic and intuitive manner. These additions have enabled CSC to achieve a final outcome displaying data in near real time whilst extending the battery lifetime of a wireless sensor network.

Contents

1 Introduction 1

1.1 Overview 1

1.2 Purpose 1

1.3 Structure & Contributions 1

1.4 Definitions, Acronyms, and Abbreviations 3

2 Requirements & Constraints 4

2.1 Mandatory Design Requirements Matrix 4

2.2 Aspirational Design Requirements Matrix 6

2.3 Project Constraints 8

3 System Architecture 9

3.1 System Transactions 11

3.2 System-Wide Payload Standard 13

3.3 Summary 15

3.3.1 Design Philosophy 15

3.3.2 Final Design Elements 15

4 Stakeholder Engagement 16

4.1 Design Review Actions 21

5 Design Decisions 22

5.1 Hardware Design 24

5.1.1 PCB Layout 24

5.1.2 Power Supply Design 24

5.1.3 Cost Analysis of PCB Fabrication 24

5.2 Embedded Design: Sensor-Host 27

5.2.1 Dusty Eterna Flash 28

5.2.2 Dusty Crystal Configuration 30

5.2.3 DuinoPro-to-Dusty Interface 32

5.2.3.1 Timing 32

5.2.3.2 Endianness 34

5.2.3.3 Interface Limits 34

5.2.4 Embedded Application Architecture 35

5.2.5 DuinoPRO State Management 36

5.2.6 Power Consumption & Management 38

5.2.7 Sleep Management 41
5.2.8 Main Mote Routine 43
5.2.8.1 Start-Up Mode 43
5.2.8.2 Scheduling Mode 43
5.2.8.3 Sampling Mode 44
5.2.9 Mote Join Routine 44
5.2.9.1 Duty Cycle Management 44
5.2.9.2 Synchronisation 45
5.2.9.3 Message Exchange 45
5.2.10 Sample – Payload Formation 46
5.2.10.1 Sample function 48
5.2.10.2 Data sampling 50
5.2.10.3 Framing 52
5.2.11 Data Acquisition 54
5.2.11.1 Timestamp 54
5.2.11.2 duinoPRO Battery 54
5.2.11.3 Sensor 54
5.2.12 Sensor Configuration 58
5.2.13 Mote Configuration 60
5.2.13.1 Configuration State Variables 60
5.2.13.2 Configuration Lookup Table 60
5.2.13.3 Configuration Set 61
5.2.13.4 Configuration Parameter Set 63
5.2.13.5 Other Configuration Functions 64
5.2.14 Network Manager and Gateway 65
5.2.14.1 Embedded Network Manager 65
5.2.14.2 Gateway Application 65
5.2.14.3 Upload Mote Data Routine 66
5.2.14.4 Update Configuration Routine 68
5.3 Front-End Design 69
5.3.1 Cloud Integration 69
5.3.2 Choice of Cloud Service Provider 69
5.3.3 Choice of Database 70
5.3.4 Database Design 71
5.3.5 Web Application Framework 74
5.3.6 Interface Design 75
5.3.7 Deployment 81

6 Testing 83
6.1 Unit Tests 83
6.2 Integration Testing 84
6.3 System testing 85
6.4 Testing Summary 86

7 Resources 87
7.1 Hardware 87
7.2 Network 87
7.3 Front End 87

8 Risks & Contingencies 88

9 Design Outputs 91

10 System Cost 92

11 Conclusion 93

12 References 94

13 Appendices 97
13.1 Appendix A – Yet to be added 97
13.2 Appendix B – IoTeam Dusty Net List 98
13.3 Appendix C – Timesheet of CSC 105

List of Figures

Figure 1: Transactions between system end-points 9
Figure 2: System architecture showing key sub-systems and interfaces 9
Figure 3: Data & Diagnostic Transaction 11
Figure 4: Configuration Transaction....12
Figure 5: Update Transaction 12
Figure 6: Payload structure and evolution from process variables. 13
Figure 7: Dataload reading procedure 14
Figure 8: Interposer Dusty PCB 24
Figure 9: PCB Cost Per Unit Produced 26
Figure 10: Functional block-diagram of Sensor-Host with payload....27
Figure 11: Eterna flash image and components....28
Figure 12: Distribution of optimal load trim values for crystal calibration over 98 samples 30
Figure 13: Frequency characterisation from LTC5800 radio generated from the 20 MHz reference....31
Figure 14: API UART pin mapping between duinoPRO and Dusty Module....32
Figure 15: Timing diagram for duinoPRO to Dusty UART communication 33
Figure 16: Timing diagram for Dusty to duinoPRO UART communication 33
Figure 17: Ripple at high UART baud rates....34
Figure 18: State control and transitions. 37
Figure 19: Sweep over frequency and network size to determine power consumption of Dusty module 39
Figure 20: Power consumption of Dusty module during packet transmission 40
Figure 21: Power consumption of Dusty module during packet reception....40
Figure 22: Power consumption of Dusty module during idle listening 40
Figure 23: Sensor-host core routine with periodic sleep....41
Figure 24: Mote Main Routine Logic....43
Figure 25: Mote Join Routine Logic....44
Figure 26: Layered approach to sampling 47
Figure 27: Sample function flowchart 49
Figure 28: Data sampling functions general structure 51
Figure 29: reserve_field function flowchart 53
Figure 30: pack_field_header function flowchart....53
Figure 31: Layered approach to sensor data reading 55
Figure 32: sensor_read function 56
Figure 33: sensor_config subroutine....59
Figure 34: Configuration functions interactions....60
Figure 35: config_set function....62

Figure 36: config_param_set function 63
Figure 37: Gateway Logic Diagram 66
Figure 38: Process and Upload Mote Data 67
Figure 39: Update Configuration Routine 68
Figure 40: Front-End Intrasystem Integration 69
Figure 41: Composite Keys 72
Figure 42: Sensor Data Table Repository 73
Figure 43: Configuration Data Table Repository 74
Figure 44: Login Screen 76
Figure 45: Login Screen Error Message 77
Figure 46: Web Application Post-Login Screen 77
Figure 47: Range Slider Illustration: Initial Selection 78
Figure 48: Range Slider Illustration: Altering displayed data 78
Figure 49: Legend Notification 79
Figure 50: Legend illustration - Both Motes selected 79
Figure 51: Legend illustration - Single Mote selected 80
Figure 52: Graph Toolbar 80
Figure 53: Set Zoom Illustration - Click and drag 80
Figure 54: Set Zoom Illustration - Result 81
Figure 55: Toggle Spikes Enabled 81

List of Tables

Table 1: CSC's Individual Contributions 1
Table 2: Mandatory Requirements 4
Table 3: Aspirational Requirements 6
Table 4: Project Constraints 8
Table 5: Final Design Elements 15
Table 6: Stakeholder Engagement Activities 16
Table 7: Design Review Actions 21
Table 8: Summary of Design Decisions 22
Table 9: Cost of PCB fabrication 25
Table 10: PCB Cost and Quantity 25
Table 11: Eterna flash components description 28
Table 12: Non-General Purpose Pin Configuration 29
Table 13: Eterna 20MHz Crystal Configuration as in Eterna Flash 30
Table 14: System states and variables. 36
Table 15: Variables used for scheduling. 41
Table 16: Sampling mode options 46
Table 17: Data sampling functions 50
Table 18: Framing layer functions 52
Table 19: Field header length indication 52
Table 20: Field header type indication 53
Table 21: CSP Weighted Decision Matrix 70
Table 22: Database Weighted Decision Matrix 71
Table 23: WAF Weighted Decision Matrix 75
Table 24: Unit Tests Summary 83
Table 25: Integration Tests Summary 84
Table 26: Integration Test Plan 84
Table 27: Development & Testing Summary 86
Table 28: Risk Ranking Matrix 88
Table 29: Risk Rank Defined 88
Table 30: Risk Register Part 1 89
Table 31: Risk Register Part 2 90
Table 32: System Cost 92

1 Introduction

The proliferation of data monitoring technologies has enabled multifarious insights into environmental conditions, system reliability, consumer behaviour, and a myriad of other fields and industries [6, 7, 8]. Sensors are one such technology, able to detect events or changes in their environment and send this information to other electronics. Energy storage poses a constraint on sensor operation in rural environments; a lack of grid connectivity exposes the importance of sagacious energy management.

1.1 Overview

Cloud Seven Consultants (CSC) has been contracted by ATAMO (the client) to investigate energy management in low-power wireless sensor networks (WSN). The project requires integration of a third party WSN with an ATAMO Arduino based development platform. Sensor data (e.g. temperature, vibration) must be measured and periodically reported over the wireless network to a cloud-based database. The firmware in the sensor must provide for reliable communications while keeping tight constraints on energy usage. This information must be accessible on the web via a graphical user interface (GUI).

1.2 Purpose

The purpose of this document is to outline CSC's final design. CSC has ascertained the proceeding information through salient stakeholder communications and consolidation of literature pertinent to the undertaking of a design project in the field of energy management in low-power WSNs.

1.3 Structure & Contributions

CSC has identified three core sub-systems within the WSN – Hardware, Kernel and Front-End – as recommended by the client [9]. Individual sub-teams have been assigned to each core sub-system. The Hardware team is tasked with designing and fabricating the interposer printed circuit board (PCB) for the duinoPRO-Dusty module. The Kernel team is tasked with developing the firmware, communication packet structure for sensor motes and optimising battery usage. The Front-End team is tasked with data visualisation via the integration of cloud services. The breakdown of the various sub-teams is illustrated in Table 1.

Table 1: CSC's Individual Contributions

Contributor	Sub-team	Content
Yung	Hardware	Hardware Design, Sensor-Host Interface, & Dusty Configuration
Aaron	Kernel	Sampling, Sensor Driver & Configuration
Jamie	Kernel	System Integration, State, Power & Sleep Management
Matt	Kernel	Mesh Network Integration and Main Routine
Peter	Kernel	Network Manager/Gateway
Andy	Front-End	Cloud Integration
Jake	Front-End	Graphical User Interface

The structure of the report has been listed below, with the team member(s) responsible for writing the section denoted in square brackets. This report has been broken down as follows:

- Section 1 introduces the project and document structure [Jake];
- Section 2 will give the final project requirements & constraints [Aaron & Jake];
- Section 3 will summarise the system architecture [Jamie];
- Section 4 will outline stakeholder engagement [Aaron];
- Section 5 will detail the final design decisions;
  - Section 5.1 will outline design decisions pertinent to the Hardware design [Jamie & Yung];
  - Section 5.2 will outline design decisions pertinent to the Embedded design [Aaron, Jamie, Matt, & Peter];
  - Section 5.3 will outline design decisions pertinent to the Front-End design [Andy & Jake];
- Section 6 will outline the testing performed on the design [Aaron];
- Section 7 will outline the project resources [Andy];
- Section 8 will outline the identification and management of safety issues, risks and contingencies [Jake];
- Section 9 will detail the design outputs [Andy];
- Section 10 will finalise and consolidate the project costs [Yung];
- Section 11 will summarise and conclude the report [Matthew];

1.4 Definitions, Acronyms, and Abbreviations

AP	Access Point
API	Application Programmers Interface
AWS	Amazon Web Services
BAU	Business as Usual
CSC	Cloud Seven Consultants
CSP	Cloud Service Provider
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HTTP	HyperText Transfer Protocol
IBM	International Business Machine
I/O	Input/Output
IP	Internet Protocol
ISR	Interrupt Service Routine
LUT	Lookup Table
LSB	Least Significant Bit
MSB	Lost Significant Bit
NRDBMS	Non-Relational Database Management System
OTAP	Over-the-air-Programming
PCB	Printed Circuit Board
RDBMS	Relational Database Management System
TCP	Transmission Control Protocol
The Client	ATAMO
UART	Universal Asynchronous Receiver/Transmitter
WAF	Web Application Framework
WSN	Wireless Sensor Network

2 Requirements & Constraints

CSC has segregated the project requirements under two headings – Mandatory and Aspirational. Mandatory requirements must be accomplished for the project to be considered a success. In this regard, there are no trade-offs between the mandatory requirements, being the baseline level of quality defined by the client. Upon realizing all mandatory requirements, CSC will pursue the delivery of additional Aspirational requirements to further client satisfaction. These requirements have either been directly spoken as desirable additional features/functionality, or have been identified by CSC as exciter requirements. These aspirational requirements pursue a more optimised system, and have inherent trade-offs due to time restrictions imposed on CSC.

2.1 Mandatory Design Requirements Matrix

The mandatory requirements have been reviewed and verified by the client through email communication [10]. The following mandatory requirements in Table 2 span across expected, spoken, and unspoken requirements.

Table 2: Mandatory Requirements

ID	Requirements	Business Need(s)
M01	Functional Operating Conditions - Operational ambient temperature: -40 to 70°C - Fully IP rated (IP68 or above) - Input Battery Voltage 3.0 to 3.6 V - Humidity: 10-90%	Continual system functionality in expected operating environment
M02	Safety: - The system must be safe to operate, install, and maintain with necessary precautions taken to ensure the system will not cause injury or damage.	Reduce danger to human life to as low as reasonably practicable
M03	Design strategy: - A clear and defined strategy for implementation of the system must be provided.	Ease of implementation, maintenance and enhancement
M04	Operational Lifetime: - Sensor devices will operate for no less than 1 year for every 1000 mAh of battery capacity supplied.	Affordability of deploying large numbers of sensors and instrumentation
M05	System Network: - The system must be capable of receiving data remotely from at least 3 sensor devices, with each device separated by a maximum of 50m in open air - The sensor hosts must be discoverable and able to synchronise to the WSN given they are within 50 m (in open air) of another SmartMesh-IP enabled host (sensor host or network manager).	Provides additional data points to alleviate domain-based errors Automates the process of data procurement
M06	Internet Connectivity: - The system must be capable of sending sensor data to a cloud-based database system from an internet-enabled gateway.	Continual data access

M07	Data validity: - Sensor data that is sent through the network (from sensor to gateway) must be valid, such that errors can be detected and stopped before transmission.	Reliable data integrity
M08	Data Specification: - Data that is transmitted from each host must contain the timestamp of the sensor data, the source of the sensor data, and the sensor data itself - Sensor data shall be obtained at maximum available precision, up to 16 bits. Hence, all sensor data transmissions shall allow for up to 16 bits of data.	Provides value where the time and location of sensor data is of practical significance
M09	Data currency and timeliness: - The user must be able to view current data from the sensors from an internet-enabled gateway with a maximum of a two (2) minute delay	Provides near real-time access to monitored data
M10	Client Application: - A clear and relevant user interface that can access and interpret the cloud-based database must be provided for demonstration purposes.	Provide convenient and coherent access to monitored data
M11	Portability: - Capable of adapting any sensor module that is designed to the ATAMO duinoPRO specification.	Provide cross-functionality across a wide range of end-uses
M12	Hardware Interface - Interface between dusty module and duinoPRO must conform to ATAMO's standard protocol	Provide cross-functionality across a wide range of end-uses

2.2 Aspirational Design Requirements Matrix

The client has reviewed the draft of the aspirational requirements in the meeting on 24/08/2017 [11] and dictated a priority rating from 1 to 5 (with 1 being of lowest importance and 5 being of highest importance). The client has verified the ranking of these requirements; the numerical priorities given by the client are listed in the column "Client Priority" (Table 3). CSC has bolstered this priority with a 'Ranking' column, which orders the aspirational requirements in descending order. Hence a 'Ranking' of one is the most important, being prioritised over all proceeding aspirational requirements.

Table 3: Aspirational Requirements

ID	Ranking	Requirements	Business Need(s)	Client Priority
A01	1	Data Specification: - In addition to sensor data of at least 16-bit resolution, diagnostic data of 16-bit resolution should be transmitted through the network	Provide critical data on the current condition of the system Accommodate planned downtime and potential maintenance strategies	5
A02	2	System Network: - The system will be capable of receiving data remotely from at least 5 sensor devices, with each device separated by a maximum of 50m in open air	Increased network capacity enables improved data resolution across the monitored site	5
A03	3	Configuration Flexibility - The system design will allow for each mote to have and receive different configurations from the network manager. Configuration parameters will include, but are not limited to, sensor sampling interval and diagnostic data sampling interval.	Ability to flexibly dispatch the network throughout a plant. Ensuring that each mote can be optimally configured to its environment and purpose	5
A04	4	Synchronise Sensor Readings - Ensure that sensor data readings occur at predictable and synchronised times at each node	Reliable and timely data acquisition	4
A05	5	Authentication - Authentication will be provided to a minimum of two users on a single-tenant platform - The users will have identical permissions	Facilitates the procurement of confidential sensor data Demonstrates the potential scalability of the system	4
A06	6	Operational Lifetime - Sensor devices will operate for no less than 2.5 years for every 1000 mAh of battery capacity supplied.	Low and infrequent system maintenance costs High reliability sensing	3

A07	7	Minimise System Cost - Ensure total system cost is no more than $60	Affordability of deploying large numbers of sensors and instrumentation	2
A08	8	Amplified User Experience - Graphical display of time series data - Consideration of best practices for Human-computer interaction; simplistic	Ability to rapidly and easily discern trends and irregularities in sensor data	2
A09	9	Virtual Manager - The system should incorporate SmartMesh IP VManager-based network management to replace local embedded-based network management - A remote x86 based virtual machine with the SmartMesh-IP VManager installed should be capable of performing network management functions when connected to a SmartMesh-to-IP gateway that is local to the WSN site	Ability to more easily scale and reconfigure network	1
A10	10	Database Scalability - The database design may be capable of accommodating for data sent by hundreds of sensors - This scalable database design should be incorporated into the cloud system with considerations for the ‘Big Data’ movement	Ability to install extensive monitoring equipment throughout a plant and manage the associated data	1

2.3 Project Constraints

CSC has identified constraints to the project that must be considered alongside the project requirements (Table 4).

Table 4: Project Constraints

ID	Constraint
C01	Sensor host application limited by duinoPRO UNO processor data memory (SRAM) of 2 Kbytes, 32 Kbytes of In-System Self-Programmable Flash memory, and 1 Kbytes of EEPROM.
C02	Interposer PCB size limited by duinoPRO module dimensions of 29.27 x 28.54 mm [12] and allowed overhang of 5 mm [13].
C03	Six GPOI pins are available for connecting the duinoPRO to module site seven. All are configurable; four are capable of acting as standard UART protocol pins [14].

3 System Architecture

This system implements a site-based WSN which uses the Transmission Control Protocol/Internet Protocol (TCP/IP) to remotely transmit data to a client/developer for analysis whilst also being capable of remote configuration by the client/developer. The three key 'end-points' in this system are shown in Figure 1.

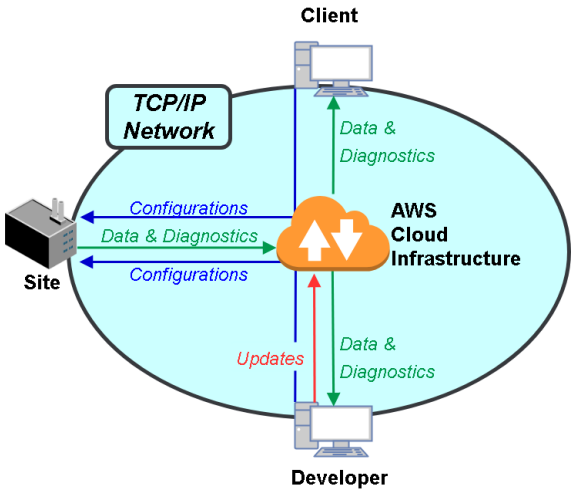


Figure 1: Transactions between system end-points

Sensor data and diagnostic information are pushed from the site to the cloud infrastructure hosted on Amazon Web Services (AWS), which provides storage and analysis services. Commands can be sent to the site to configure sensor-hosts and updates can be patched by developers for the cloud services. Figure 2 provides a more detailed view, exposing key sub-systems required for these transactions.

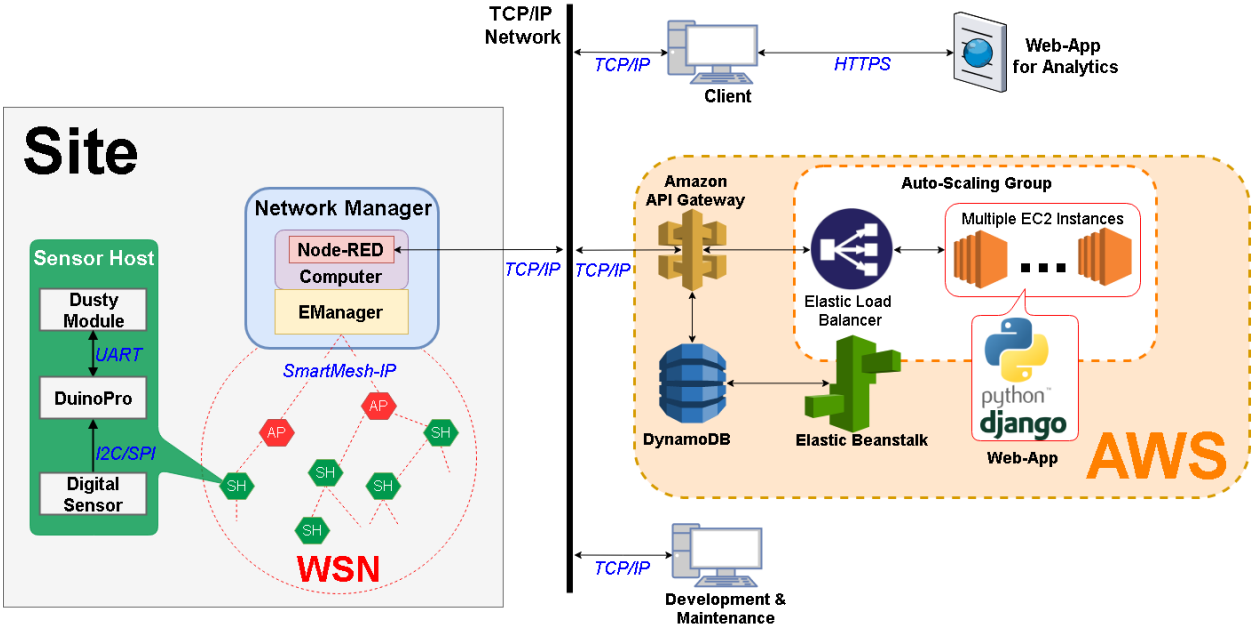


Figure 2: System architecture showing key sub-systems and interfaces

As seen in Figure 2, sensor-hosts (SH) and access-points (AP) are placed within the site to create a self-forming

multi-hop mesh WSN. The sensor-hosts within the WSN are AVR-based microcontrollers (duinoPRO) with digital sensors and Dusty modules which provide communication using Linear Technology's SmartMesh-IP network protocol. Sensor-hosts sample data and diagnostics, and transmit/forward the resulting payload to the centralised network manager node. The network manager uploads this data using the Node-RED service to AWS.

The sub-systems in AWS integrate with one another to perform two key tasks:

1) Parse data sent from the WSN and store it in a structured format for querying, and
2) Serve data and controls through a web-application accessible through a HTTP-browser.

The web-application accesses the cloud database, DynamoDB, to process the data for analysis and visualisation. Further, it exposes interfaces that allow users to configure the WSN remotely. The same payload structure is used for this task (Section 3.2), with each configuration parameter represented as one field. The web-application is designed with Python Django and orchestrated with Elastic Beanstalk. Orchestration in this context refers to the automatic-deployment and automatic-scaling of infrastructure to serve the web-application to clients. This includes virtual servers for hosting the application, load balancing, and the DynamoDB interface for querying. Additionally, Elastic Beanstalk provides a deployment service that allows developers to rapidly patch the web-application. Finally, the Amazon API Gateway service provides a means of authentication and interprets and directs in/out-bound streams.

3.1 System Transactions

From a transactional viewpoint, as seen in Figure 1, there are 3 key transaction types within the system:

1. **Data & Diagnostics**: process variables are sampled from site and sent upstream to the relevant end points (Figure 3)
    a. *Site-to-AWS*: Raw data (process variable) is sent up-stream for storage and analysis
    b. *AWS-to-Client/Developer*: Processed data is provided to the end-users
2. **Configurations:** Clients/developers can send configurations downstream to configure individual sensor-hosts on site (Figure 4).
3. **Updates**: Developers can patch the AWS infrastructure downstream (Figure 5).

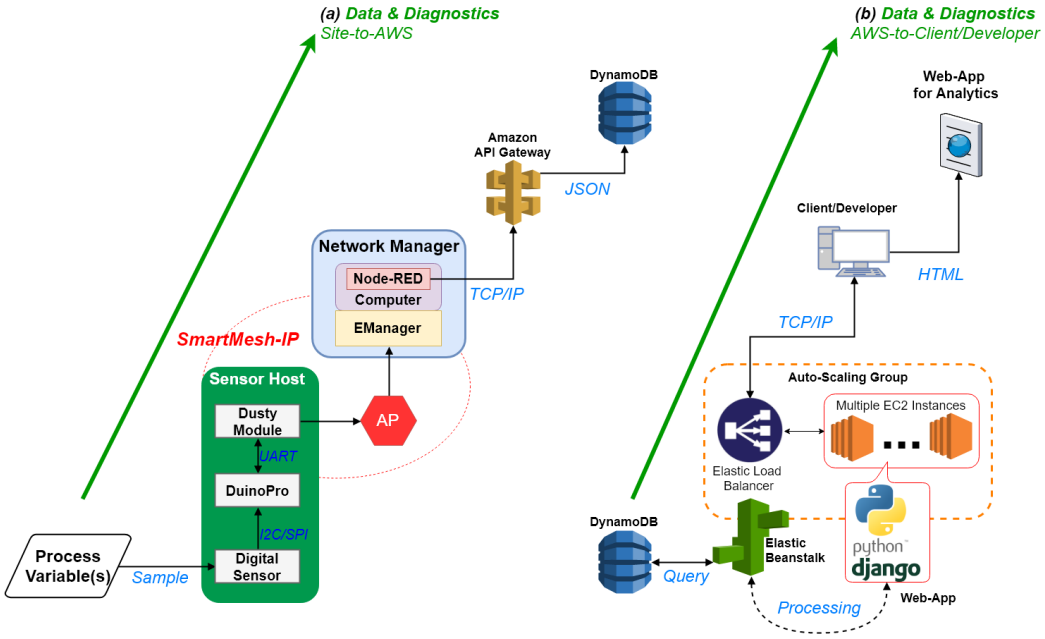


Figure 3: Data & Diagnostic Transaction

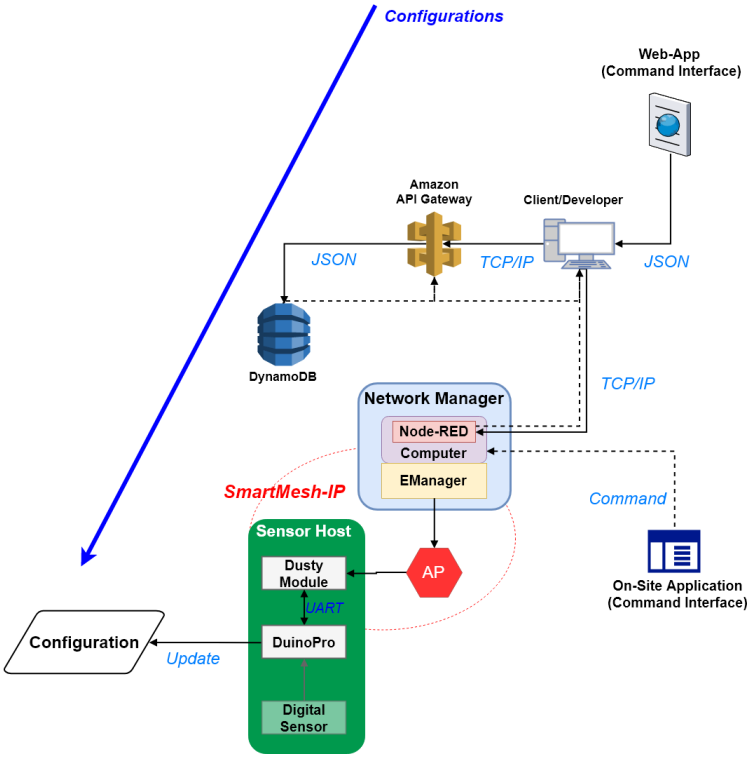


Figure 4: Configuration Transaction

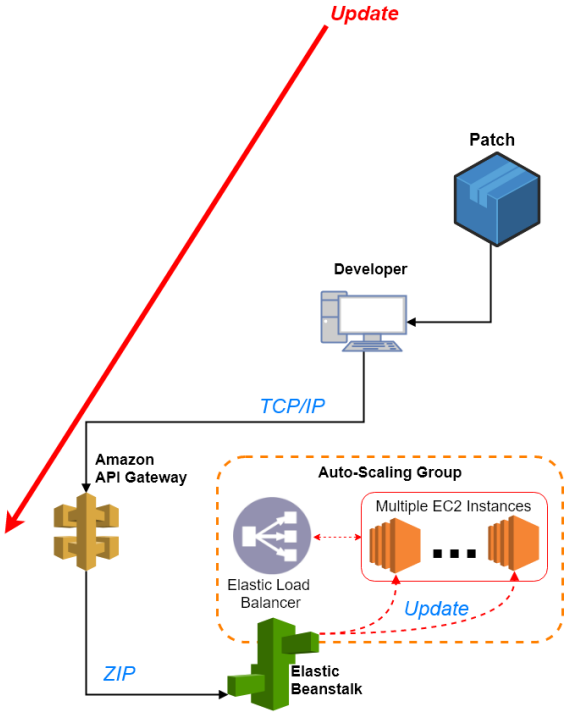


Figure 5: Update Transaction

3.2 System-Wide Payload Standard

The payload standard that is sent throughout the network is shown in Figure 6

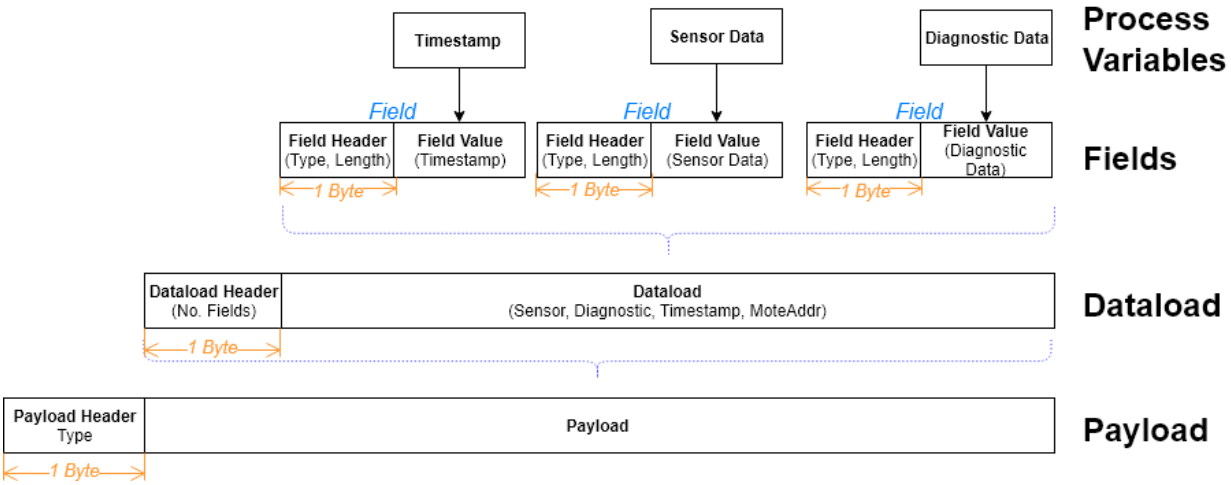


Figure 6: Payload structure and evolution from process variables.

The purpose of this payload structure is to ensure scalability and portability. Fields of information can easily be added and removed without affecting the payload standard. The use of headers assists all end-points in analysing the information; allowing devices to prepare buffers for the incoming payload/dataload.

As seen in Figure 6, the payload begins its assembly from 'process variables' – the information to be communicated. These are called 'fields' (as in 'fields of information'). Each process variable (the *field value*) is preceded by a 1 byte "Field Header" which describes what the field type is (i.e. the type of information) and length of the field value (in bytes). Note that the data in the *field value* is stored most significant byte first.

Multiple fields are then packed and wrapped into a *dataload*. The dataload has a single 1 byte header which details how many fields are expected in the payload. This serves the purpose of informing functions which disassemble the payload how many times the 'field' decomposition process should be performed.

Finally, the dataload is wrapped into the payload. The payload header is crucial in informing recipients the type of payload to expect. This is critically important to distinguish between Data & Diagnostic transactions and Configuration transactions.

This structure of headers allows designers to implement a simply disassembly routine as seen in Figure 7.

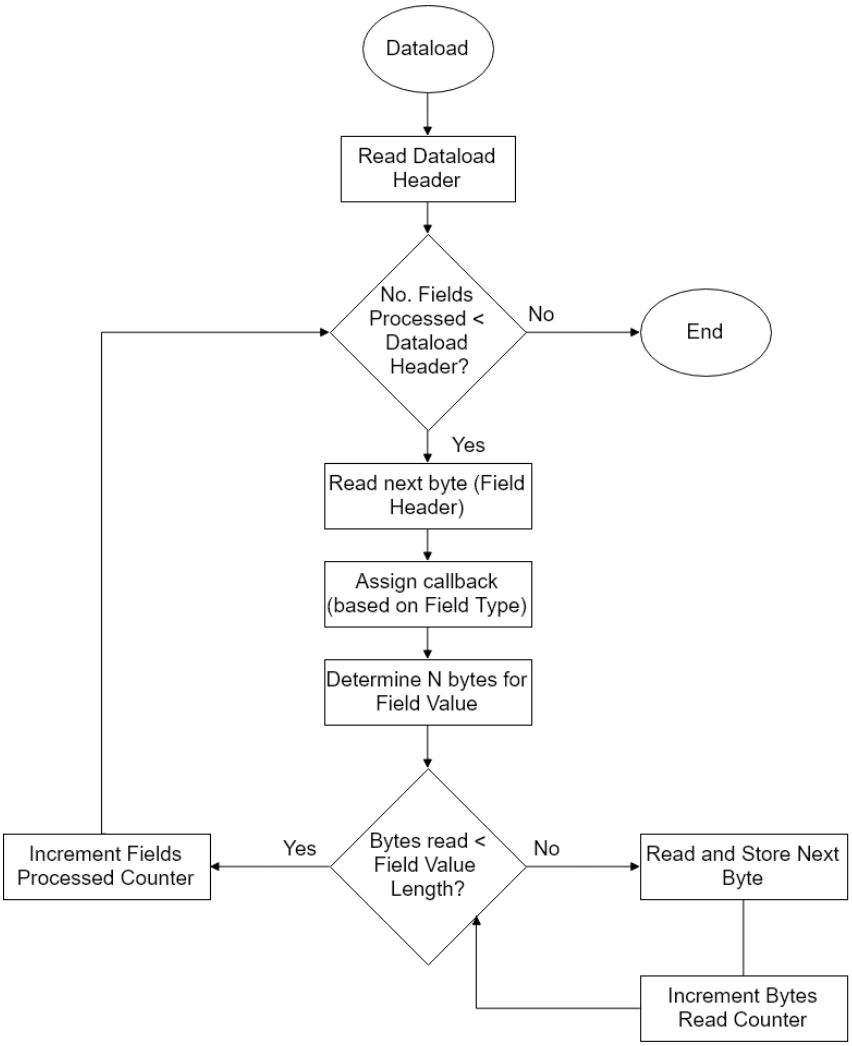


Figure 7: Dataload reading procedure

3.3 Summary

This section provides a brief summary of the design philosophy and outputs.

3.3.1 Design Philosophy

All design decisions were regulated to ensure the continual alignment of the project scope with the client's expectations. Each sub-team initially prioritised the implementation of the mandatory requirements specified in Table 2. Only after basic functionality of a component was achieved and the relevant mandatory requirements were met did CSC extend the design to meet the aspirational requirements outlined in Table 3. This enabled the team to optimise the man-hours assigned to the project, maximising high-value work aligned with the client's preferences. A critical path analysis was regularly undertaken to ensure calibration between sub-teams, ensuring all deadlines would be met according to schedule.

3.3.2 Final Design Elements

The final design elements (outputs) are listed in Table 5 below.

Table 5: Final Design Elements

Element	Description	Location
Requirements Analysis	Defined the system requirements	LMS
Preliminary Design	Recorded first-pass design work	LMS
System Design Strategy	The architectural document that forms the basis for the design.	This document
User Manual	Manual for users and developers to assist in the understanding of the system	https://CloudSevenConsulting.github.io/
Testing Document	A detailed description of testing conducted on the system throughout the design process (from design, implementation and deployment)	LMS
Preliminary Implementation	CSC's preliminary implementation of the system	https://github.com/CloudSevenConsulting

4 Stakeholder Engagement

Key stakeholder engagement activities undertaken during this project are outlined in Table 6 below. For each activity, the relevant issues and decisions are described briefly alongside the resulting action items. Unless otherwise noted, all actions are for CSC. Stakeholder names are abbreviated within the table as: SM (Dr Sally Male), MC (Mr Mark Callaghan) and MP (Mr Marcus Pham).

Table 6: Stakeholder Engagement Activities

Date	Event	Stakeholders	Issues & Decisions	Actions
31/07/2017	Project brief released	SM MC	❖ Design elements include: third party WSN, ATAMO development platform, cloud-based database. ❖ Design considerations include: reliable communications and energy usage.	❖ Research design elements and considerations
07/08/2017	Project outline received from client	MC	❖ Additional design elements: PCB for WSN module, network manager/gateway and data display interface. ❖ Emphasis placed on sleep management of each device to optimise energy usage. ❖ Confidentiality & IP status clarified.	❖ Continue research
10/08/2017	Project Partner Meeting	MC MP	❖ Emphasis placed on layered software approach with power management as a layer. ❖ Sampling rate, operating environment, network size, device current draw, system lifetime, cost and database security specified. ❖ Design documentation with clear approach considered most important requirement.	❖ Continue research ❖ Develop draft requirements ❖ Form sub-teams
17/08/2017	Technical Queries 1-6	MC	❖ System architecture, operating conditions assumptions and draft requirements approved. ❖ Expected system lifetime is 1 year/1000 mAh. ❖ Packet structure will require bytes indicating the type of data/packet. ❖ Allow for 16-bit sensor data.	

Date	Event	Stakeholders	Issues & Decisions	Actions
24/08/2017	Project Partner Meeting	MC MP	❖ Allowable data acquisition delay is 2 minutes. ❖ Cost requirements clarified. ❖ The ability to vary sampling rate is desirable.	❖ Construct hardware testing setup for duinoPRO and Dusty ❖ Complete Requirements Analysis
25/08/2017	Requirements Analysis shared with client	MC		
06/09/2017	Technical Queries 7-8	MC	❖ Interposer PCB only required for motes, not APs or network manager. ❖ Allowed overhang of 3-5 mm suggested for PCB. ❖ Dusty module will be programmed before assembly.	
06/09/2017	AWS access received	MC	❖ Username, password and setup information received.	
07/09/2017	Project Partner Meeting	MC MP	❖ Database and GUI technology selected. ❖ Web-hosted GUI preferred. ❖ Sampling rate configuration for motes clarified (default, plus ability to update from network manager). ❖ Use of a scheduler such as Guarded Atomic Actions recommended. ❖ Clarified function of duinoPRO pins.	❖ Update mote main routine in light of new configuration requirement ❖ Dusty-duinoPRO pin mapping with pin descriptions ❖ MC to provide duinoPRO libraries and GAA resources
08/09/2017	duinoPRO libraries received, feedback on minutes	MC	❖ Default mote configuration will be compiled in with updated values stored in RAM. ❖ Motes should be inactive if not connected to a network.	

Date	Event	Stakeholders	Issues & Decisions	Actions
21/09/2017	Preliminary Design Interviews & Review Meeting	MC MP SM	❖ Requirements for hardware documentation clarified (pins, communications). ❖ ISRs for hardware communication must be brief. ❖ Sensor resolution clarified to be variable, but 16-bits must be allowed for. ❖ Configuration data should be stored by mote in the database, not timestamp. ❖ Desirable GUI features include: zoom, selecting start time and different display intervals.	❖ Complete prototype system for software & hardware testing. ❖ Describe reason for inclusion or non-inclusion of each pin in Dusty-duinoPRO pin mapping ❖ Timing diagram for UART ❖ Investigate PCB volume pricing ❖ Clarify how timestamps are collected and used ❖ Investigate minimisation of communication from gateway to minimise cost ❖ Address how configuration data will be handled in the cloud database ❖ Implement additional GUI features

Date	Event	Stakeholders	Issues & Decisions	Actions
28/09/2017	Design Review Meeting	MC MP	❖ Decision to only design the Dusty PCB and not attempt to complete layout and fabrication (time constraints). ❖ Django (for GUI) noted as being suitable for final design ❖ UART mode 2 confirmed as more desirable due to its more robust handshake protocol. ❖ If multiple interrupts are used, timing could become an issue for simultaneous interrupts. ❖ State management of motes from network manager should be considered. ❖ In the database, configuration data should be stored by mote ID. ❖ It is desirable for GUI to be scalable to large numbers of motes/sensors. ❖ Limited memory of the ATMega328P processor flagged as an issue.	❖ Investigate PCB test points ❖ Note option to select mote sensor driver at compile time ❖ Provide clear definition of payload and dataload headers in packet structure definition ❖ Define sleep management process during network joining ❖ Assess interrupt latency (if multiple ISRs are used) ❖ Clarify when network time is queried in program ❖ Develop plan for mote state tracking at the network manager ❖ Change primary key for configuration table to composite key (mote ID). ❖ Develop plan for displaying large numbers of sensors in GUI ❖ Record the choice of software licence in report. ❖ Complete code documentation (using DOxygen)
29/09/2017	Emails regarding GitHub and Linear Technology account	MC	❖ GitHub links and MyLinear account details shared with MC.	

Date	Event	Stakeholders	Issues & Decisions	Actions
07/10/2017	Email regarding reorganisation of GitHub repositories	MC	❖ New GitHub links and structure shared with MC.	
15/10/2017	AWS Access Incident	MC	❖ AWS access credentials were inadvertently pushed to the GitHub (publicly accessable), but quickly removed and reset. Amazon detected the release of information and MC was immediately notified.	❖ Team rules put in place for avoiding and handling similar situations in future.
19/10/2017	Project Partner Meeting	MC MP	❖ UART issues discussed: messages sent correctly, but actions/responses not occurring. ❖ Duty-cycle management in mote join routine discussed and considered non-mandatory. ❖ Decision to cease development and dedicate time to reporting.	❖ Provide a single link/document that provides access to all project documentation, code, info, etc. ❖ Provide instructions for setting up database.

4.1 Design Review Actions

Specific actions resulting from the design review meeting are repeated below in Table 7 along with the sections which contain the responses to these actions.

Table 7: Design Review Actions

Action	Section Addressing Action
Investigate PCB test points	13.2
Note option to select mote sensor driver at compile time	5.2.12
Provide clear definition of payload and dataload headers in packet structure definition	3.2
Define sleep management process during network joining	5.2.9
Assess interrupt latency (if multiple ISRs are used)	Multiple ISRs not required
Clarify when network time is queried in program	5.2.10
Develop plan for mote state tracking at the network manager	5.2.14
Change primary key for configuration table to composite key (mote ID).	5.3.4
Develop plan for displaying large numbers of sensors in GUI	5.3.6**Error! Reference source not found.**
Record the choice of software licence in report.	5
Complete code documentation (using DOxygen)	See User Manual submission

5 Design Decisions

The various design decisions have been summarised in Table 8. The "Justification" column points to the section in which the analysis for the design decision has been conducted.

Table 8: Summary of Design Decisions

Design Decision	Selection	Justification	Relevant Requirements & Constraints
Source Code Licensing	Open Source / MIT	Promotes use of system, client preference [15]	
Power supply output impedance	Limit output impedance of power supply to 5Ω	5.1.2	
Radio Reference Crystal Oscillator Trim Value	Trim 20 MHz crystal by 64 ppm	5.2.2	M05, A02
Endianness for multi-byte UART	Little-Endian	5.2.3.2	
UART Baud rate	9600 bits per second	5.2.3.3	
UART maximum frame length	8 bytes	5.2.3.3	
UART maximum message queue	4 messages	5.2.3.3	
Maximum sampling rate	Maximum sampling rate of 0.1 Hz (1 sample per 10 seconds)	5.2.6	M04
Sleep-guard Lock Prevention	Configurable timeout for sleep-guard protection in scheduling	5.2.5	M04, A06
Sampling	Layered and approach to sampling-related functions	5.2.10	M07, M08, A01
Read sensor data	Layered and approach to sensor data reading functions	5.2.11.3	M07, M08, M11
System parameters identification in configuration payloads	One-to-one match between field type in configuration payloads, index in device configuration array and lookup table	5.2.13.2	A03, C01
duinoPRO baseboard	duinoPRO UNO	Availability	
Hardware Communication Protocol	UART Mode 2	5.2.3	M07, M09
PCB fabrication vendor and cost	EasyEDA	5.1.3	A07

Design Decision	Selection	Justification	Relevant Requirements & Constraints
Duty Cycle Management	Dynamically Update	5.2.9	M04
Choice of Network Manager	Embedded Manager	5.2.14	M05, A02
Choice of Cloud Service Provider	AWS	5.3.2	M03, M08, M09, A07
Choice of Database	Amazon DynamoDB	5.3.3	M03, M08, M09, A03, A10
Database Design	Sensor Data Table & Configuration Data Table [Partition Key: MoteID] [Sort Key: Timestamps]	5.3.4	M03, M08, M09, A03, A07, A10
Web Application Framework	Django	5.3.5	M03, M10, A08,
Interface Design	Sign in authentication; interactive GUI displaying time-series data	5.3.6	M03, M10, A05, A08, A10
Deployment	AWS Elastic Beanstalk	5.3.7	M03, M10, A08, A10

5.1 Hardware Design

The hardware design is concerned with coupling between two modules: the Dusty module provided by IoTeam, and the duinoPRO development board provided by ATAMO. This section covers the design of the interposer PCB required to adapt the Dusty module to the duinoPRO. To facilitate the transfer of this data between the Dusty module and duinoPRO, CSC has chosen API UART Mode 2 as the hardware communication protocol (see Section 5.2.3). CSC has also recommended EasyEDA as the PCB vendor to fabricate the PCB.

5.1.1 PCB Layout

given the hardware communication protocol, the interposer Dusty PCB can be designed via KICAD. The board layout of the interposer Dusty PCB encompasses a Surface Mount Device (SMD) which adapts to the Dusty module and a series of castellated mounting holes across the edge of the board as shown in Figure 8. Furthermore, unused pins of the Dusty required proper pin termination to ensure Dusty operates as intended. A detailed description of pin mapping and termination is presented in Appendix B – IoTeam Dusty Net List.

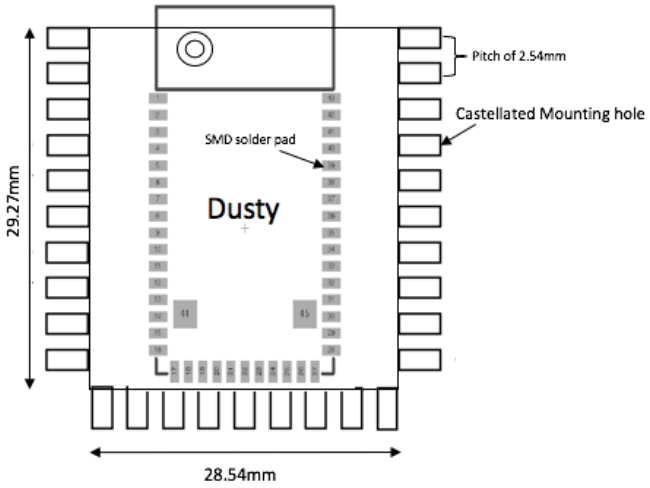


Figure 8: Interposer Dusty PCB

5.1.2 Power Supply Design

Due to its heavy duty cycling for radio communication, the Dusty module's current consumption has a substantial demand profile. It is recommended that a power supply of low impedance ($Z_{DC,Out} < 5\ \Omega$) [16] be used such that it can respond to sudden changes in current consumption [16].

From the current profiles in Figure 20, Figure 21, and Figure 22, the power supply for the Dusty module must be able to ramp from $250\ \mu A$ to $10\ mA$ in less than $1\ \mu s$ without generating a transient voltage greater than $200\ mV$ co-incident with the current ramp. This ramp is seen prominently during preparation of transmission for data transmission ('Tx Prep' in Figure 20) and sending the acknowledge byte after data reception ('Tx Acknowledge' in Figure 21).

5.1.3 Cost Analysis of PCB Fabrication

In order to satisfy requirement A07, CSC has conducted cost analysis of PCB fabrication for the interposer Dusty PCB from two PCB vendors, namely Wurth Elektronik and EasyEDA. Table 9 lists the fabrication, shipping, tax and total cost for a single PCB for these vendors [16, 17].

Table 9: Cost of PCB fabrication

PCB Vendor	Wurth Elektronik	EasyEDA
Cost of fabrication	$75.65	$4.93
Cost of shipping	$10.45	$28.88
Tax	$16.35	-
Total Cost	**$102.45**	**$33.81**

As seen from Table 1, the total cost of EasyEDA is cheaper than Wurth Elektronik. However, Wurth offers services such as PCB routing verification, troubleshooting and additional PCB features, whereas EasyEDA only offers basic services. However, for the purposes of this project, EasyEDA is the most preferred vendor as it able to fulfil the project requirements.

In addition to the above, CSC investigated how the pricing of PCB fabrication varies with quantity. This will be of interest to ATAMO should this system be brought into large-scale production. As shown in Table 10 and Figure 9, larger quantities correspond to a lower per-unit price.

Table 10: PCB Cost and Quantity

PCB Quantity	Cost of Single PCB	Total Cost of PCB Fabrication
50	$0.89	$44.33
100	$0.50	$49.92
150	$0.34	$51.06
200	$0.27	$54.34
250	$0.23	$57.63
300	$0.20	$61.07
350	$0.18	$64.36
400	$0.17	$67.64
450	$0.16	$70.93
500	$0.15	$74.37

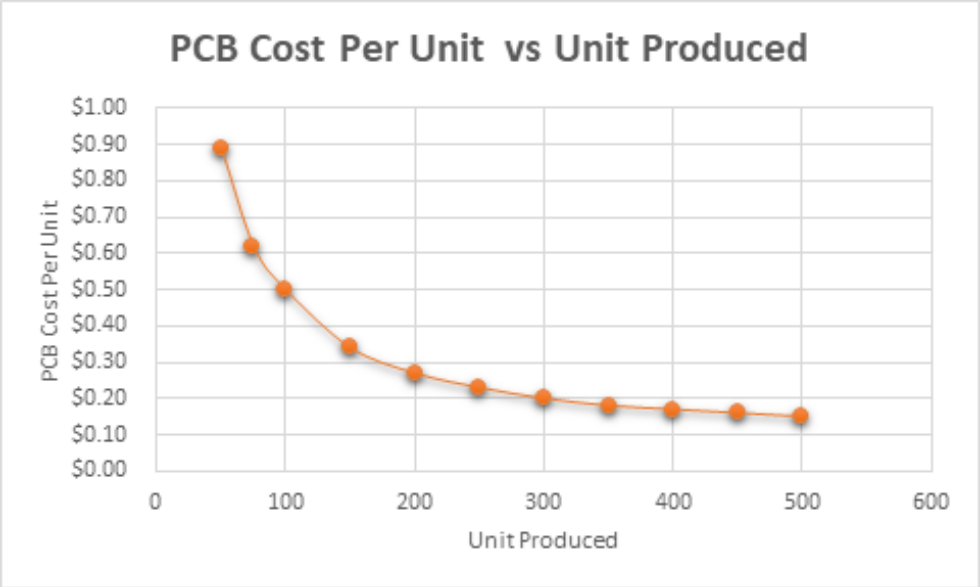


Figure 9: PCB Cost Per Unit Produced

5.2 Embedded Design: Sensor-Host

Sensor-hosts form the nodes within the system network described in requirement requirements M05/A02. They are responsible for sensing physical data (process variables) from a site and transmitting the data to a central node, as specified in requirements M07, M08, and M09. The implementation of the sensor-host by CSC consists of three key devices:

1. The IoTeam Dusty Module, which uses the LTC5800-IPM (Eterna) System-on-Chip,
2. The UNO duinoPRO platform based on Atmel's ATmega328P microcontroller, and
3. A sensor module occupying one of six module spaces on the duinoPRO platform.

The embedded application of the sensor-hosts refers to the firmware, logic, and drivers of the duinoPRO platform. The embedded application is crucial in addressing requirement M04/A06 (through scheduling strategies), A03 (using the application layer), and A04 (using the network layer).

Figure 10 shows these three devices as well as their core functional blocks, key units and interfaces between them.

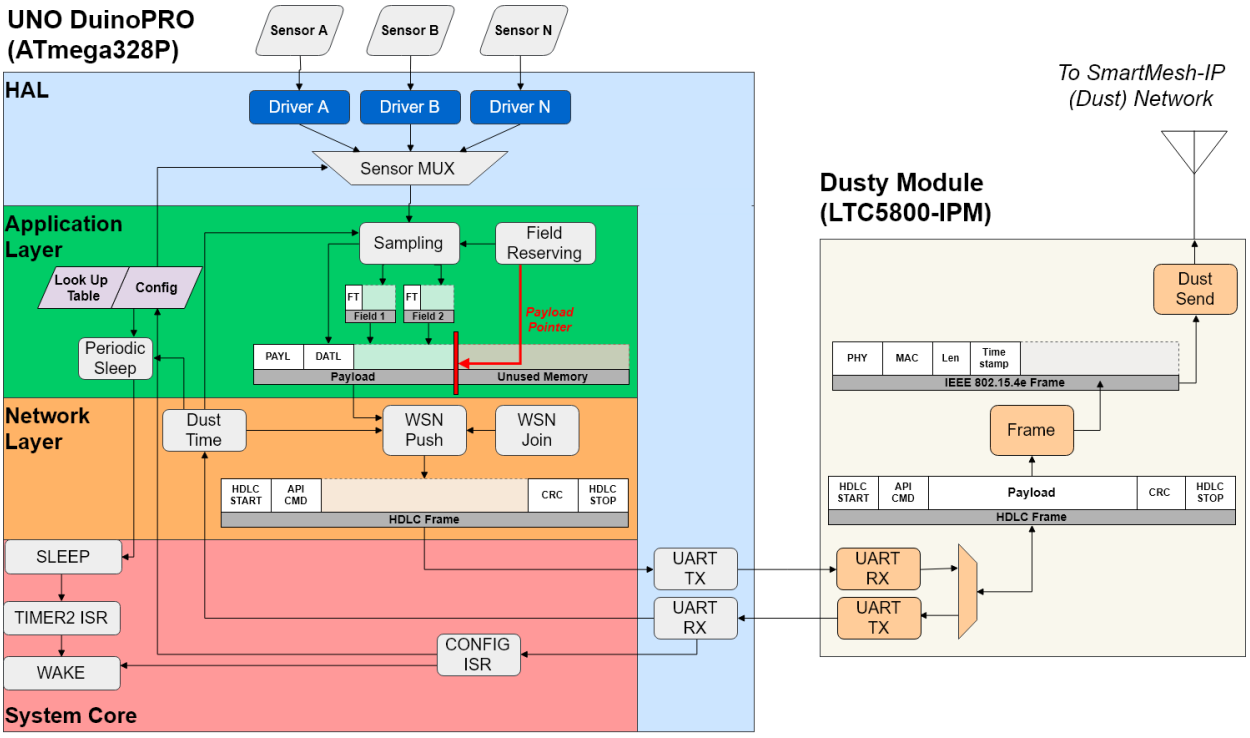


Figure 10: Functional block-diagram of Sensor-Host with payload

5.2.1 Dusty Eterna Flash

In order to integrate the Dusty module with the sensor-host, the Dusty module must have a custom embedded flash program based on Linear Technology's Eterna System-on-Chip solution (herein after called the 'flash program'). The flash program is 512 kB in size and divided into 4 image components as seen in Figure 11 and Table 11 [17].

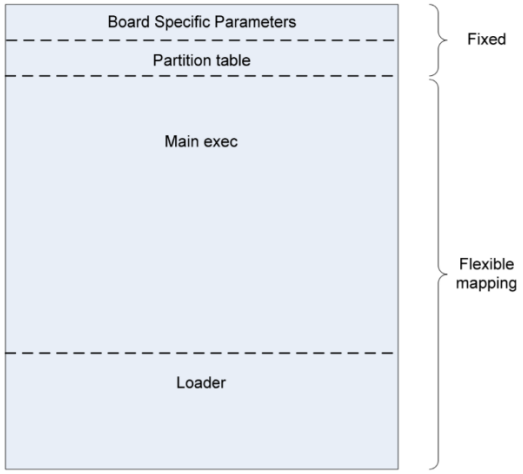


Figure 11: Eterna flash image and components

A key configuration for the Dusty module is setting the device into 'Slave mode', in which it forfeits control of its radio functionality to an external processor which communicates through its API or CLI communication ports. Whilst the Dusty Module is a System-on-Chip solution which can act as its own master, CSC elected to use an external master microprocessor (the duinoPRO) to satisfy requirement M11.

Table 11: Eterna flash components description

Image Component	Start Address (Hex)	Description
Fuse Table	0	2 kB image containing hardware and software configuration settings.
Partition Table	800	Defines the location of the elements in the Flexible mapping portion of Eterna's image
Main Executable	1000	The main executable image. Each variant of the Eterna product family has a corresponding main executable
Loader	77800	The loader manages handling of completed Over-the-air-Programming (OTAP) images and starting the Main Executable image.

Importantly for these design, the Fuse Table must be configured correctly to ensure operation of the Dusty device. Through this configuration, CSC has elected to disable the majority of the Dusty's Input/Output (I/O) functionality to minimize power consumption as per requirement M04/A06. The remaining non-general purpose pin functionality is listed in Table 12**Error! Reference source not found.**.

Table 12: Non-General Purpose Pin Configuration

Pin #	Signal	Direction	Pull	Power Consumption
22	RESETn	Input	Pull-Up	50 nA
23	TDI	Input	Pull-Up	50 nA
24	TDO	Output		
25	TMS	Input	Pull-Up	50 nA
26	TCK	Input	Pull-Down	50 nA
37	UARTC0_TX	Output		
38	UARTC0_RX	Input	Pull-Up	50 nA
55	FLASH_P_ENn	Input	Pull-Up	50 nA
66	UART_RX_RTSn	Input		
67	UART_RX_CTSn	Output		
68	UART_RX	Input		
69	UART_TX_RTSn	Output		
70	UART_TX_CTSn	Input		
71	UART_TX	Output		

5.2.2 Dusty Crystal Configuration

A 20 MHz crystal oscillator source (OSC_20M) is used as the radio frequency reference for the LTC5800 System-on-Chip and is critical for radio communication [16], and hence requirement M05/A02. Because of the criticality of this component, Linear Technology has recommended selection of this component from one of the performance verified vendors. IoTeam has elected the ECS crystal for their Dusty module as listed in Table 13.

Table 13: Eterna 20MHz Crystal Configuration as in Eterna Flash

OSC_20M Crystal Part Number	ECS-200-CDX-0914
OSC_20M Load Trim	64 ppm

The 20 MHz crystal has 2 capacitors which support its functionality, known as load capacitors; the LTC5800-IPM System-on-Chip has these capacitors in-built [17]. The values of the load capacitors is controlled through the fuse table at memory address 0x13, the 'Load Trim' setting [17]. Load trimming is done to account for variations in printed-circuit board layout and dielectric stack-up. The extent of trimming is determined through frequency characterisation performed on a statistically meaningful number of targets to account for unit-to-unit variation.

The frequency characterisation process begins by using a calibrated timing reference signal from Linear Technology's DC9010 to characterise the target board layout and return an optimal load trim (or pull value) [17]. Over a sample of boards, the optimal trim value is then used and verification of the frequency characterisation is performed to determine if the trim value is adequate for the design. CSC has determined the optimal value for the evaluation Dusty module to be 64 ppm; however, CSC recommends that future designers re-perform this analysis for new systems over a more statistically significant sample size.

As an example, the distribution of recommended trim values based on the DC9010's calibrated timing signal over a set of boards is seen in Figure 12 [17].

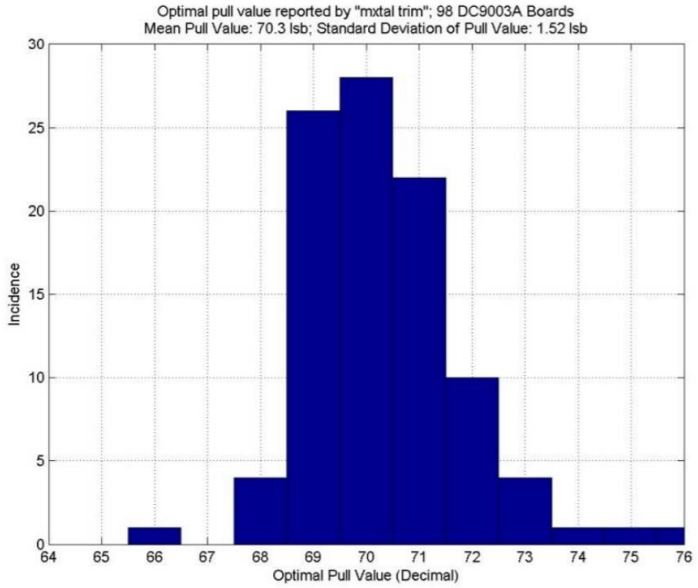


Figure 12: Distribution of optimal load trim values for crystal calibration over 98 samples

From this, the boards are then configured with the optimal trim value (in this case 70 ppm) and frequency characterisation is performed with the DC9010 to determine the spread of frequency errors as seen in Figure 13 [17].

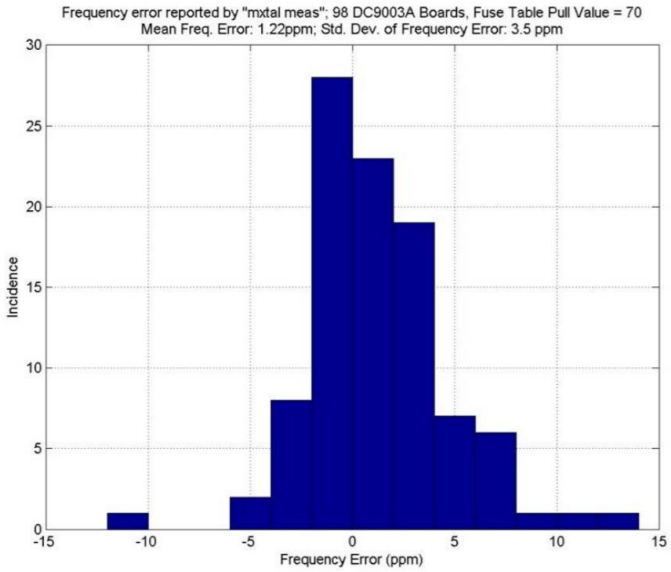


Figure 13: Frequency characterisation from LTC5800 radio generated from the 20 MHz reference

The histogram in Figure 13 shows a well-centred, normally-distributed set of errors which are largely caused from the integer rounding in the load trim value set in the flash program's fuse table. Such a histogram profile suggests the load trim value is well suited for the board and conditions. However, this analysis must be re-performed for changing conditions (most notably in this case, temperature).

5.2.3 DuinoPro-to-Dusty Interface

The duinoPRO and Dusty module on-board the sensor-host communicate with one another through the serial Universal Asynchronous Receiver/Transmitter (UART) protocol. This is the only available protocol in which the Dusty module exposes for master control through an external processor, in this case the duinoPRO.

There are two distinct ports in which UART master control, the API UART and the CLI UART. The CLI UART serves as a debugging port in which a Serial Terminal installed on a computer can communicate to the Dusty module with command line interfaces. The API port is designed such that a set of programming interfaces can be used by a microcontroller, such as the duinoPRO, without the need for a command line interface. Hence the API UART port is the designated communication protocol between the duinoPRO and Dusty module.

There are two options for API UART, mode 2 (with 6 pins) and mode 4 (with 4 pins). CSC has elected to use mode 2 as this mode includes two additional control flow signals (along the duinoPRO-to-Dusty lines) for more robust communication; mode 4 is constrained to a smaller operating temperature range due to reliability issues in flow control [18]. CSC has determined that mode 4 would compromise requirements M07 and M09 due to this lack of reliability.

The 6 pins of API UART mode 2 is mapped to the duinoPRO as seen in Figure 14**Error! Reference source not found.**.

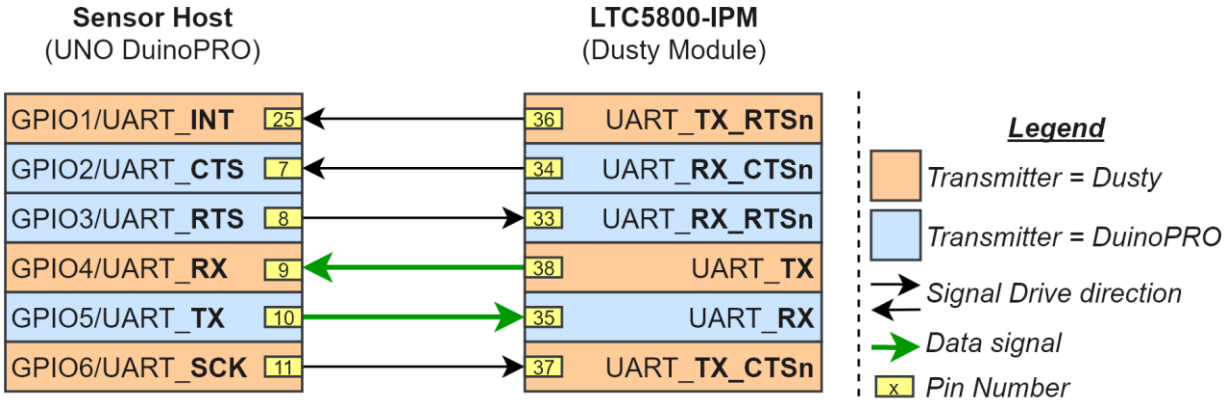


Figure 14: API UART pin mapping between duinoPRO and Dusty Module

5.2.3.1 Timing

The process of sending data from the duinoPRO to the Dusty module, as shown in Figure 15, is as follows;

1. The duinoPRO begins communication by asserting (falling-edge) the RTS (UART_RTS → UART_RX_RTSn) line.
2. The Dusty module responds (as an acknowledgement) by asserting (falling-edge) the CTS (UART_CTS ← UART_RX_CTSn) line
3. After the duinoPRO detects the asserted CTS line, it transmits the HDLC framed payload.
4. Following the transmission of the final byte in the HDLC payload, the duinoPRO negates (rising-edge) the RTS line.
5. The Dusty module responds by negating (rising-edge) the CTS line
6. Communication can again restart after a designated time period of $t_{interbyte}$.

The entire communication process takes a net time as shown below

$$t_{sh-dusty} = 2 \cdot t_{RTS-CTS} + t_{CTS-RX} + t_{RX-CTS} + (N_{byte} - 1) \cdot t_{interbyte}$$
$$= 2(2\text{ms}) + 20\text{ms} + 22\text{ms} + (N_{byte} - 1) \cdot 100\text{ms}$$

Formally, the time (in milliseconds) to transmit data from duinoPRO to dusty as a function of the length of the payload is then:

$$t_{DP-dusty}(N) = 100(N-1) + 46$$

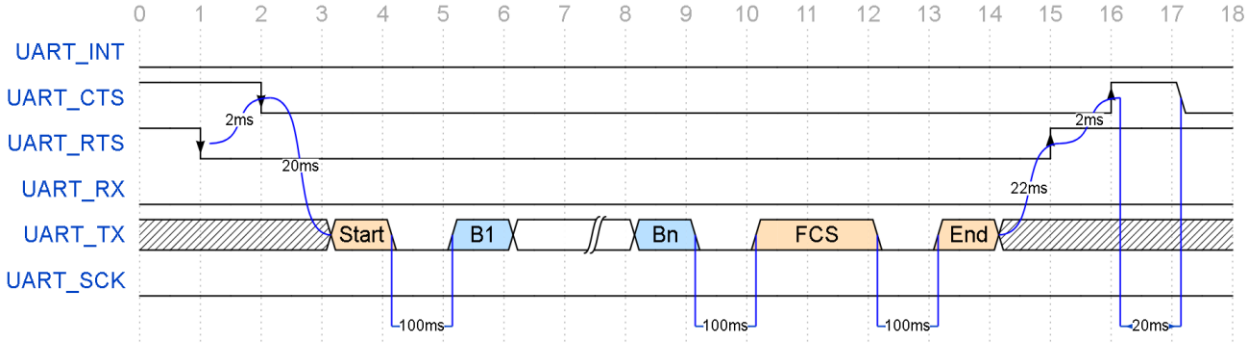


Figure 15: Timing diagram for duinoPRO to Dusty UART communication

The process of sending data from the the Dusty module to the duinoPRO, as shown in Figure 16, is as follows;

1. The Dusty module begins communication by asserting (falling-edge) the TX RTS (UART_TX_RTSs → UART_INT) line.
2. This triggers an interrupt in the duinoPRO, to which it responds (as an acknowledgement) by asserting (falling-edge) the CTS (UART_CTS → UART_TX_CTSn) line
3. The Dusty module then transmits the HDLC framed payload to the duinoPRO
4. Following the transmission of the final byte in the HDLC payload, the Dusty module negates (rising-edge) the RTS line.
5. The duinoPRO responds by negating (rising-edge) the CTS line
6. Communication can again restart after a designated time period of $t_{interbyte}$.

The entire communication process takes a net time as shown below

$$t_{sh-dusty} = 2 \cdot t_{RTS-CTS} + t_{CTS-RX} + t_{RX-CTS} + (N_{byte} - 1) \cdot t_{interbyte}$$
$$= 2(2\text{ms}) + 20\text{ms} + 22\text{ms} + (N_{byte} - 1) \cdot 100\text{ms}$$

Formally, the time (in milliseconds) to transmit data from duinoPRO to dusty as a function of the length of the payload is then:

$$t_{DP-dusty}(N) = 100(N-1) + 46$$

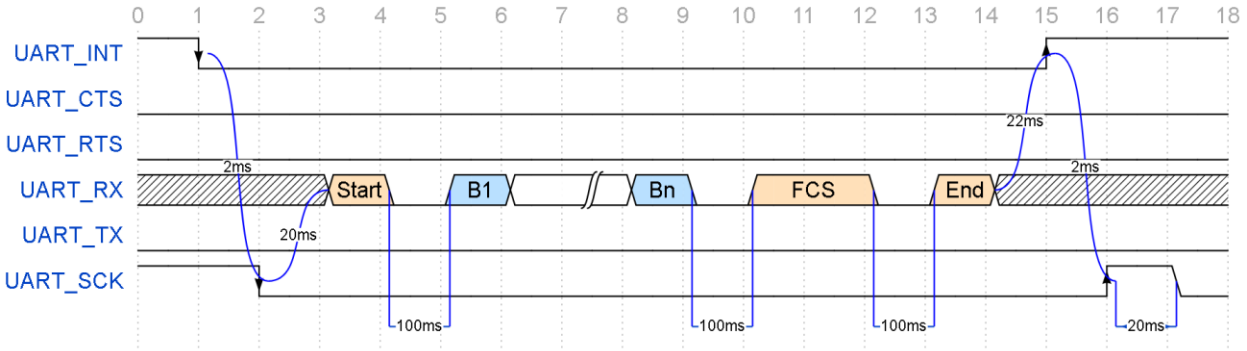


Figure 16: Timing diagram for Dusty to duinoPRO UART communication

5.2.3.2 Endianness

The ATmega328P which the DuinoPro is based on is a Little-Endian platform, and hence the UART module ports little-endian functions when accessing greater than 1-word memory.

5.2.3.3 Interface Limits

The UART interface is limited with a maximum payload of 8 bytes. This is to ensure enough data memory can be reserved for other units of functionality in the embedded application. 8 bytes was specifically chosen, as this is the expected limit for 'business-as-usual'; with 3 bytes for the sensor field, 3 bytes for the diagnostic field, and 2 bytes for the timestamp field.

The UART is designed with a message buffer to allow the slower of the two devices time to process incoming messages. This buffer is also restricted to a maximum of 4 queued messages (each of maximum 8 bytes). This is primarily motivated to reduce data memory consumption on the device. As per constraint C01, the DuinoPro device only has 2KB of SRAM, in which must be utilized for higher priority functionality.

The baud rate of the interface is set to 9600 bps, as opposed to the optional 115.2 kbps. This is due a (relatively) low frequency ripple that occurs at higher transmission frequencies. As seen in Figure 17**Error! Reference source not found.**, a ripple at 48.2% of the signal can be observed when the baud rate is set to 115.2 kbps.

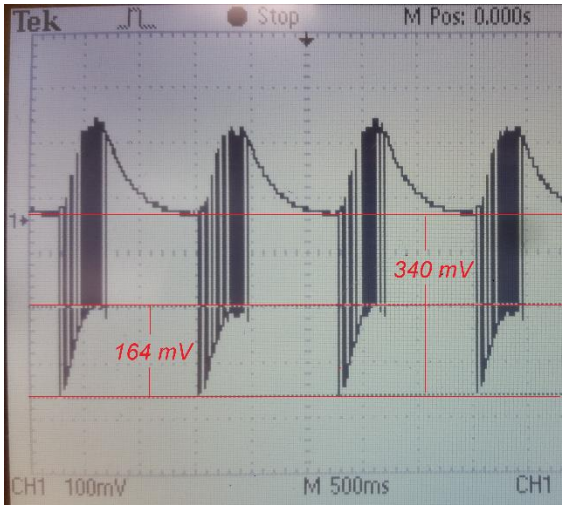


Figure 17: Ripple at high UART baud rates

5.2.4 Embedded Application Architecture

As required by M03, CSC has designed a layered-model of the embedded application for the duinoPRO with 4 layers, these are, from lowest abstraction to highest:

- System-Core
- Hardware Abstraction Layer (HAL)
- Network Layer
- Application Layer

The system-core refers to the internal abstraction of hardware; it is the interface to system-calls such as power management, and timing.

Alongside the system-core layer, the HAL is the lowest-level layer as it is *implementation dependent*. The HAL provides an interface to the device's hardware, and ensures that the implementation details of the interface remain hidden (or abstracted) from the other layers. This ensures that when the hardware implementation changes, as it does with the sensor drivers and sensor MUX, the layers that interface the HAL do not need to change, as they remain indifferent to the implementation. This is consistent with requirement M11.

The network-layer is a middleware layer; one that allows the integration of different systems. In this case, this middleware layer integrates the DuinoPRO with the Dusty module. It provides the API functional C-port which other layers can reliably call.

The application layer is the highest-level layer as it provides the core functionality of the embedded-application. It is responsible for determining what and when to sample data, implementing the payload structure as described in section 3.2, and scheduling and managing its utility and memory.

5.2.5 DuinoPRO State Management

The key state variables that determine the sensor-host's states, and the corresponding states are shown in Table 14. These states are determined from the required operation of the sensor-host. The "Wake-up" and "Low-Power" states are consequences of M04/A06; the "Processing" and "Interrupt" states provide for the processes that implement data sampling as required by M07, M09, and A04 and configuration flexibility (A03); the "Initialization" and "Joining" states are required for the transient periods in which the system first starts or when the network fails to connect the device. To maintain safety of the system (M02), an "Error" state is provided for system faults in which are marked as critical.

As mentioned Table 14 the state variables will be encapsulated, in which only the processes that invoke state changes can access. Exposing functions will be provided by these processes to allow for state changes.

Table 14: System states and variables.

State	Ready	Configuration Loaded	Connected	Interrupt Flags	Blocking	Sleep Guard
Initialization	0	0	0	0	0	1
Error State	0	X	X	X	X	0
Wake-up	0	1	X	1	1	1
Low-Power	1	1	X	0	0	0
Joining	1	1	0	X	1	1
Processing	1	1	1	0	X	1
Interrupt	1	1	1	1	1	1

The processes that invoke state changes are shown in Figure 18, in which either an event or process will have a corresponding function associated to it. These functions will access the globally shared state variables and modify them as required.

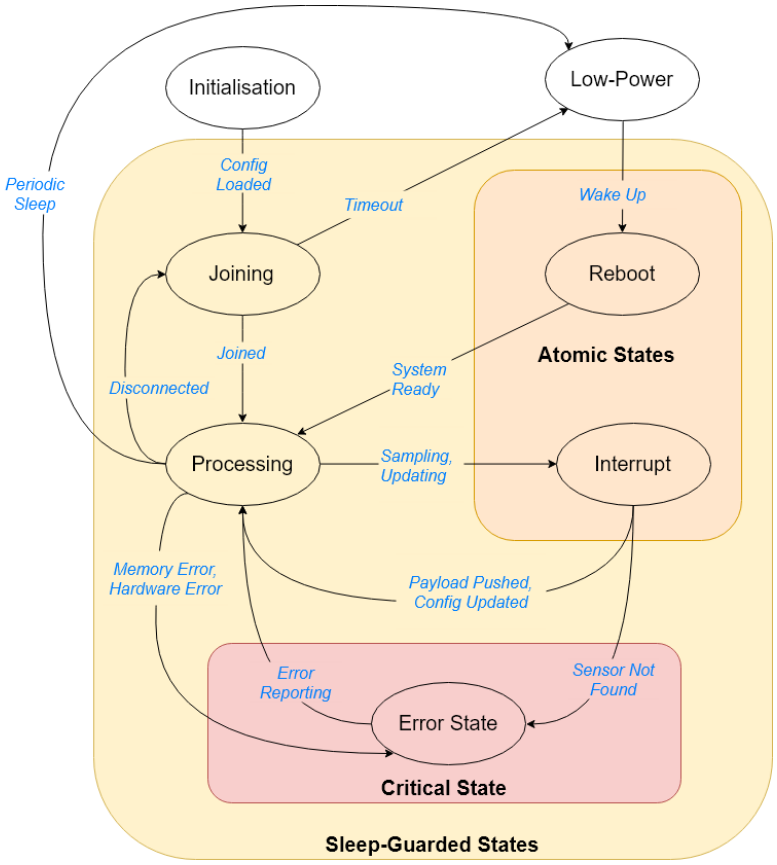


Figure 18: State control and transitions.

5.2.6 Power Consumption & Management

The Dusty module performs three atomic operations throughout its lifetime in this system, each of which have distinct power demand profiles; transmission of packets (Figure 20, Figure 21 and Figure 22) and idle listening for packets (Figure 22) [16] These atomic operations only occur during the Dusty's scheduled time-slot as determined in the Dust SmartMesh-IP protocol. Outside of the scheduled time-slots, the Dusty module is in its 'Doze' state with its lowest power consumption. These atomic operations only occur during the Dusty's scheduled time-slot as determined in the Dust SmartMesh-IP protocol. Outside of the scheduled time-slots, the Dusty module is in its 'Doze' state with its lowest power consumption.

During a scheduled transmission of sensor data, the Dusty module performs an atomic transmission and reception – The atomic transmission consumes a maximum charge of $Q_{Dust,TX} = 54.5\ \mu C$ over $7.25\ ms$ (as seen in Figure 20: Power consumption of Dusty module during packet transmission), whilst the atomic reception consumes a significantly smaller maximum $Q_{Dust,RX} = 32.6 \mu C$ of charge over $7.25\ ms$ (as seen in Figure 21). If the sample rate of the sensor-host is $F_{SH,sample}$, and $T_{SH,sample} = \frac{1}{F_{SH,sample}} \gg 7.25\ ms$, then the average power consumption due to packet transmission of the sensor-host's own packets ($\bar{P}_{Dust,TX}$) is:

$$\bar{P}_{Dust,TX} = \left(Q_{Dust,TX} + Q_{Dust,RX}\right) \cdot F_{SH,sample}$$

The Dusty module must also forward packets from neighbouring motes in the mesh network. This is the largest consumption of power for the Dusty module. Each forward operation requires an atomic idle listen, consuming $Q_{Dust,Idle} = 6.4$ µC of charge every 7.25 ms (Figure 22), an atomic receive, and an atomic transmit. For a network of N motes, if we assume an average packet forwarding rate of $\eta_{FWD} \bar{P}_{Dust,FWD}$) is:

$$\bar{P}_{Dust,FWD} = \left(Q_{Dust,TX} + Q_{Dust,RX} + Q_{Dust,idle}\right) \cdot F_{SH,sample} \cdot \eta_{FWD} \cdot N$$

In addition to this, the net leakage current from the Dusty's pins, as seen in Figure 22, consume approximately $300\ nA$ at $3.6\ V$, thus the net idle power of the Dusty module, $P_{Dust,idle}$ is given as:

$$P_{Dust,idle} = 1.08\ uW$$

Thus, the average net power consumption of the Dusty module ($\bar{P}_{Dust,net}$) is given by:

$$\bar{P}_{Dust,net} = P_{Dust,idle} + \bar{P}_{Dust,TX} + \bar{P}_{Dust,FWD}$$

CSC has determined that for a network of 50 motes sampling every 10 seconds with a packet forwarding rate of $\eta_{FWD} = 50\%$ the Dusty consumes a maximum current of $244\ uW$, 59% of the prescribed energy allowance in requirement M04. Whilst this calculation is well beyond the network requirements specified in M05, and is an overstatement of the power consumption, it provides a useful baseline for designing the sensor-host scheduling system.

Figure 19 shows $\bar{P}_{Dust,net}$ over the 2-D space sweeping along the sensor-host sampling rate, $F_{SH,sample}$, and the network size, N.

At this extremum, the duinoPRO must utilize 41% of the prescribed energy allowance in requirement M04, or an average power consumption of $170\ uW$.

With the duinoPRO having an active power consumption of $I_{DP,active} = 0.2\ mA$ and a power-saving consumption of $I_{DP,save} = 0.75\ \mu A$ [19] the net power consumption of the duinoPRO is then:

$$\bar{P}_{DP,net} = V_{dd}\left(I_{DP,active} \cdot \eta_{DP} + I_{DP,save}(1 - \eta_{DP})\right)$$

Where:

- $\bar{P}_{DP,net}$ is the net average power consumption of the duinoPRO
- V_{dd} is the supply voltage
- η_{DP} is the utilization of the duinoPRO

Solving for the utilization;

$$\eta_{DP} = \frac{\bar{P}_{DP,net}}{V_{dd}\left(I_{DP,active} - I_{DP,save}\right)} - \frac{I_{DP,save}}{I_{DP,active} - I_{DP,save}}$$

For a target $\bar{P}_{DP,net} = 170\ \mu W$ the duinoPRO must have a maximum utilization of $\eta_{DP} \leq 23.3\%$ to satisfy requirement M04.

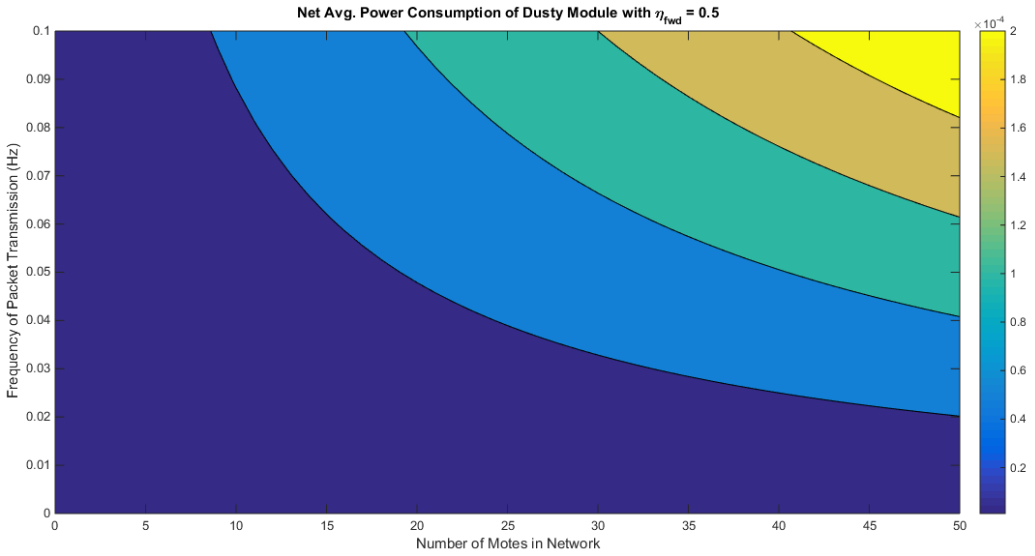


Figure 19: Sweep over frequency and network size to determine power consumption of Dusty module

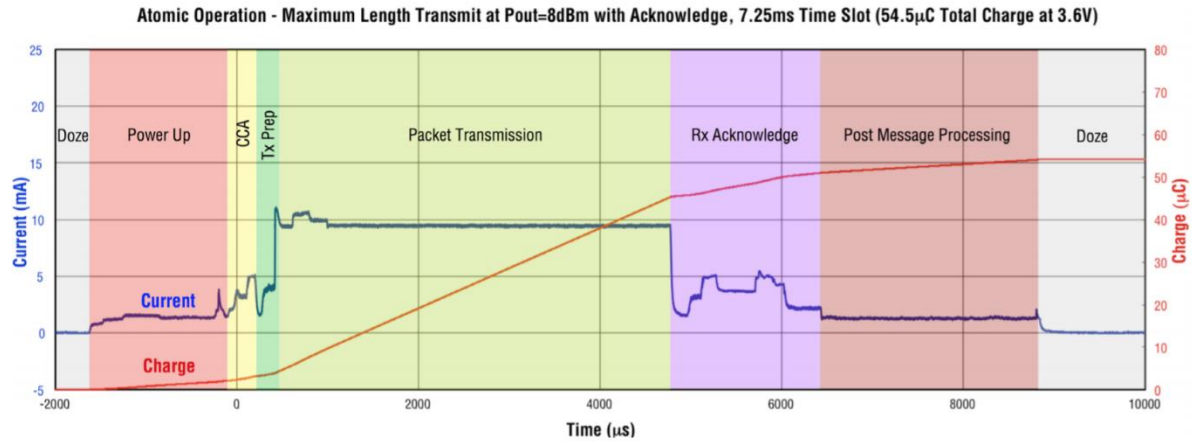


Figure 20: Power consumption of Dusty module during packet transmission

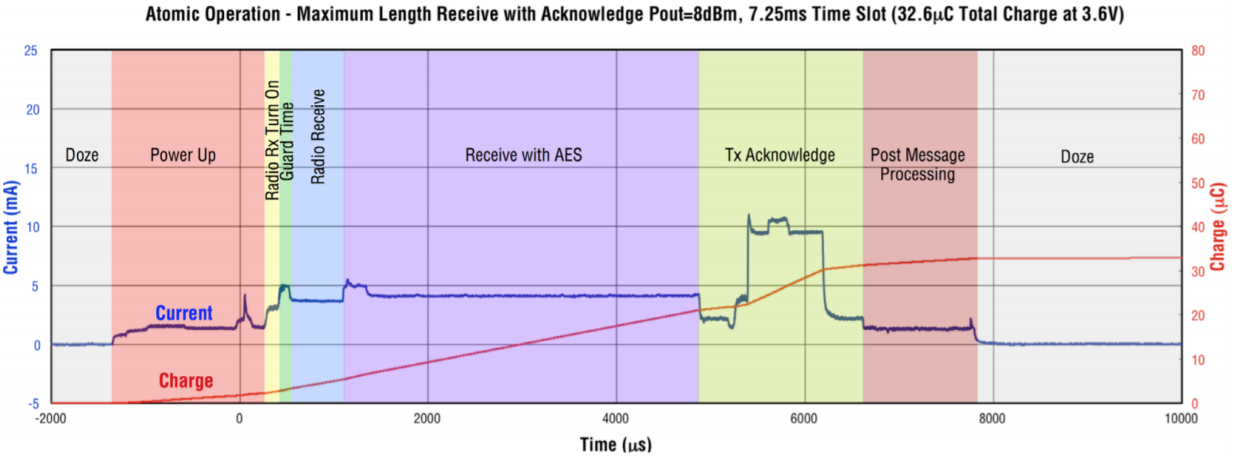


Figure 21: Power consumption of Dusty module during packet reception

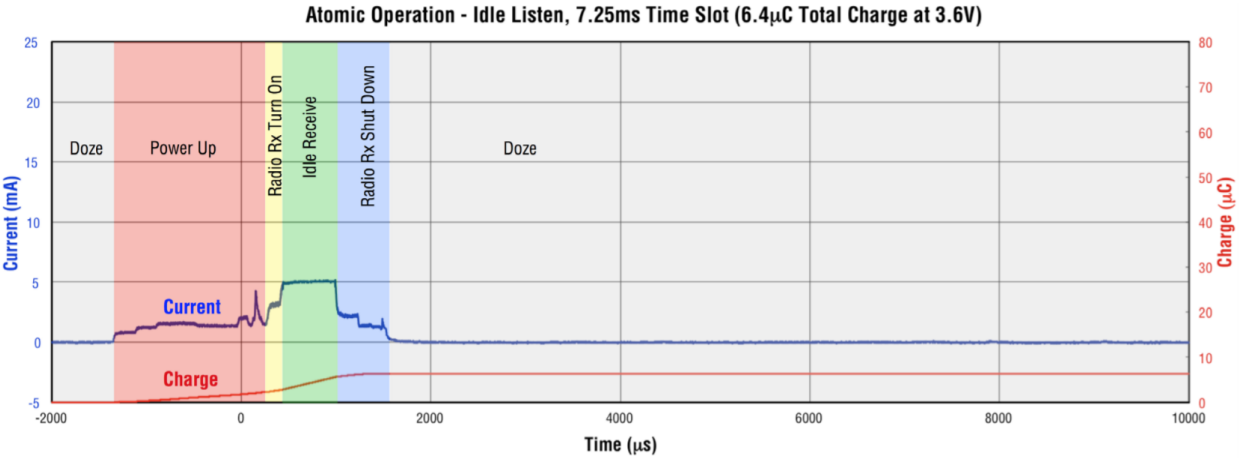


Figure 22: Power consumption of Dusty module during idle listening

5.2.7 Sleep Management

With a maximum utilization of $\eta_{DP} \leq 23.3\%$ as described in section 0, CSC has implemented an effective scheduling regime which ensures the duinoPRO device is only active when it needs to be. This sleep schedule can be seen in Figure 23 with the key variables described in Table 15.

Table 15: Variables used for scheduling.

Parameter	Description	Calculation
T_{now}	The current time	From Dust Network clock
$T_{next-sample}$	The timestamp for the next sample	Calculated from sampling rate configuration
$T_{overhead}$	A configuration parameter which accounts for errors in sleeping time, and allows overhead time for rebooting	Given from configuration
$T_{AVR,sleep}$	The time the sensor-host is in low-power state as determined by:	$T_{AVR,sleep} = (T_{now} - T_{next-sample}) - T_{overhead}$
$T_{AVR,wait}$	The time the sensor-host must wait to sample data at the correct time	$T_{AVR,sleep} = (T_{now} - T_{next-sample})$

The core routine depicted in Figure 23 shows the timeline of the duinoPRO's device state during one sample period. The net active time is then determined by:

$$t_{DP,active} \leq \eta_{DP} \cdot T_{SH,sample}$$

Similarly, the net low-power and reboot time is given by:

$$t_{DP,save} \geq (1 - \eta_{DP}) \cdot T_{SH,sample}$$

There are however constrains on $t_{DP,active}$, as the duinoPRO requires a minimum time for the system to perform processing, wake, and sleep commands.

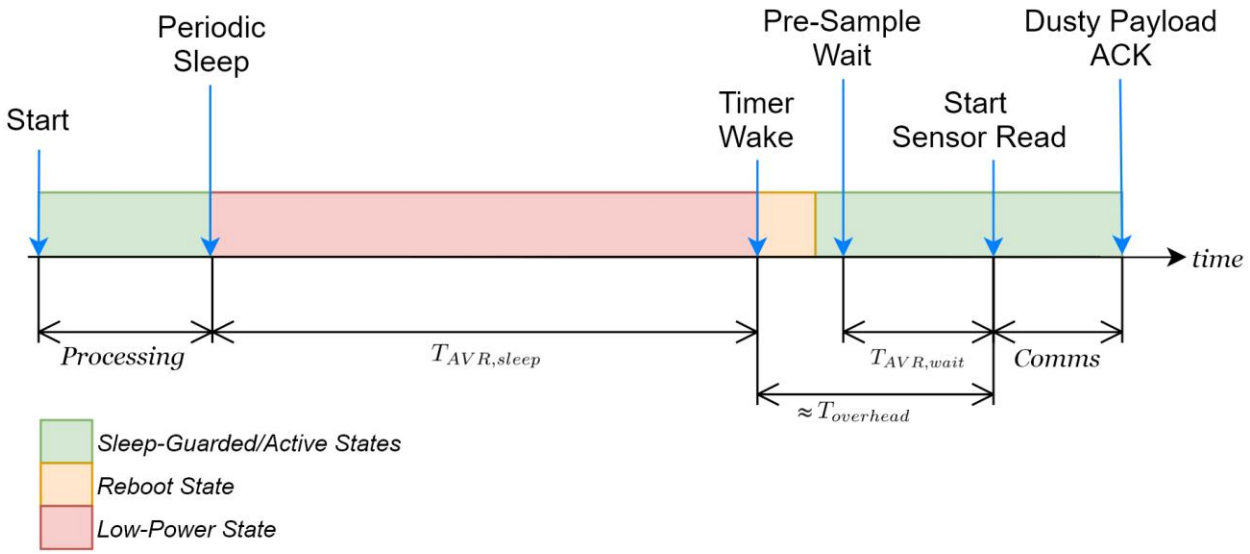


Figure 23: Sensor-host core routine with periodic sleep

If we are to assume $t_{DP-dusty}(N) \gg t_{DP,wake} \gg t_{DP,sleep}$ and the maximum payload is 16 bytes then the duinoPRO must be active for at least the period in which the duinoPRO is communicating to the Dusty module

$$t_{DP,active} \geq t_{DP-dusty}(16) = 100(16-1) + 46\,ms = 1.55\,seconds$$

Where:

- $t_{DP-dusty}(N)$ is defined in Section 5.2.6.

Thus the maximum sensor host sampling frequency is constrained such that:

$$\frac{t_{DP,active}}{\eta_{DP}} \geq T_{SH,sample} = \frac{1}{F_{SH,sample}}$$

$$F_{SH,sample} \leq \frac{\eta_{DP}}{t_{DP,active}}$$

For $t_{DP,active} = 1.55\,seconds$:

$$F_{SH,sample} \leq \frac{0.233}{1.55} = 0.15\,Hz$$

This is already in excess of the maximum frequency of $0.1\,Hz$ CSC has designed for in ensuring requirement M04 is met through the Dusty module power consumption in section 0.

If more aggressive power saving is employed (at the risk of data loss) by forcing the duinoPRO to sleep during Dusty communication time then we relax the assumption $t_{DP-dusty}(N) \gg t_{DP,wake} \gg t_{DP,sleep}$, then:

$$t_{DP,active} \approx t_{DP,wake}$$

The wake time is dictated by the start-up time of the crystal oscillator for the ATmega328P's system clock. This is driven by the fact that any clock source requires a sufficient V_{dd} to start oscillating correctly, along with a minimum number of oscillating cycles before it can be considered stable [19].

During start-up the ATmega328P issues an internal reset with a time-out delay ($T_{AVR,TOUT}$). The time-out delay allows sufficient time for a minimum V_{dd} to reach its steady state value on turn-on. Thus, $T_{AVR,TOUT}$ must be set larger than the V_{dd} rise time [19].

Following this, the clock oscillator is required to oscillate a minimum number of cycles before it is considered stable by the ATmega328P [19].

If we use the maximum $T_{AVR,TOUT} = 69\,ms$ and the maximum clock cycles of $N_{clk} = 258\,K + 258\,CK = 516\,CK$, with a system clock frequency of $12\,MHz$;

$$t_{DP,active} \approx t_{DP,wake} = 69\,ms + \frac{516}{12\,MHz} = 69.04\,ms$$

Thus;

$$F_{SH,sample} \leq \frac{\eta_{DP}}{t_{DP,active}} = 3.37\,Hz$$

Which again is within the tolerable levels to which CSC has designed for.

5.2.8 *Main Mote Routine*

The main mote routine is the main driver of the wireless network system. It is responsible for determining the state of the duinoPRO, Dusty module and sensors. The mote is capable of reading configuration files from the network manager and returning the sensor readings at the appropriate intervals. It achieves this through calls to individual subroutines and a complex scheduling system as seen in Figure 24.

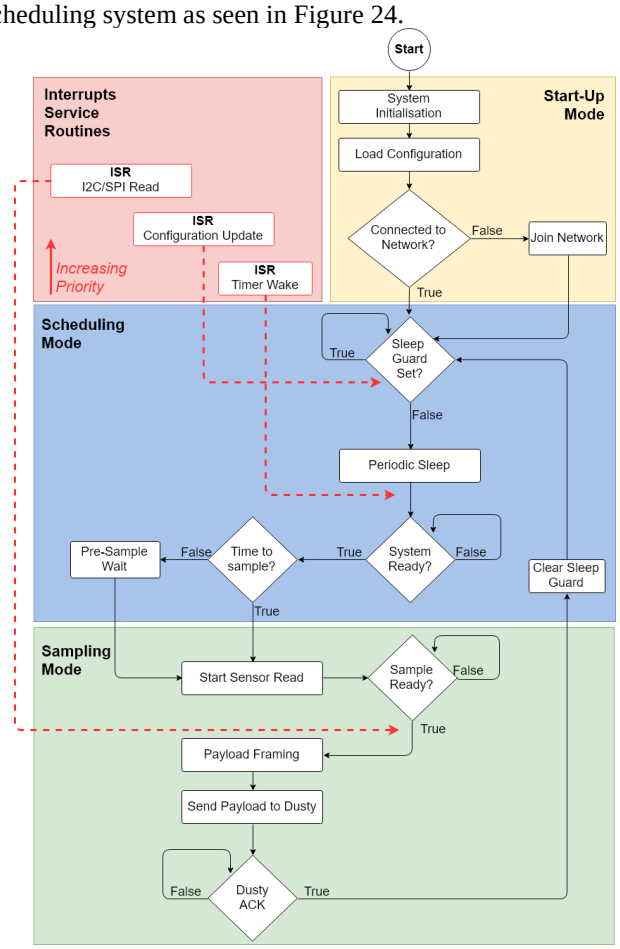


Figure 24: Mote Main Routine Logic

5.2.8.1 *Start-Up Mode*

System initialisation of the mote is largely autonomic within the SmartMesh API [4]. Initial system checks are run on both the duinoPRO and Dusty on start-up which are largely not user-configurable [4]. After the mote is initialised, it will load the default configuration parameters which will be hardcoded into the main routine. If the system is already connected to a network, it will proceed to scheduling mode. If not, the mote will call the “Network Join” How? subroutine specified in Section 3.2 before entering scheduling mode.

5.2.8.2 *Scheduling Mode*

The scheduling mode of the main routine ensures that the sensor readings are taken at the correct time intervals as specified by requirement A03. When a user alters the desired configuration parameters, the network manager will interrupt this scheduling routine and update the parameters which are stored as a global variable in the main routine, enabling requirement A04 to be met. The mote will then return to sleep and wait for the system to be ready to enter sampling mode.

5.2.8.3 Sampling Mode

Initially the mote will drive the sensor to take a reading by calling the "sample" subroutine outlined in Section 5.2.10**Error! Reference source not found.**. This subroutine writes a complete payload ready for sending to the Dusty module for transmission in the WSN, ensuring requirements M07, M08 and A01 are met. The payload is then sent to the Dusty module, ready for detection by the network manager. Once the payload has been dispatched, an automated acknowledgement signal is sent back to the mote [5]. Upon receipt of the acknowledgement signal or an ISR resulting from a new user-generated configuration file, control of the mote is handed back to the scheduling mode.

5.2.9 Mote Join Routine

Figure 25 visualises the join routine responsible for connecting each individual mote to the network. All motes will need to run the routine the first time they are powered up, and anytime they lose connection to the network in order to satisfy requirement M05. Motes do not leave the network when they enter a scheduled sleep state [6]. This routine can take anywhere from twenty seconds to tens of minutes, hence the power management of the mote was the main driver of the design in order to meet requirements M04 and A06 [2, 7].

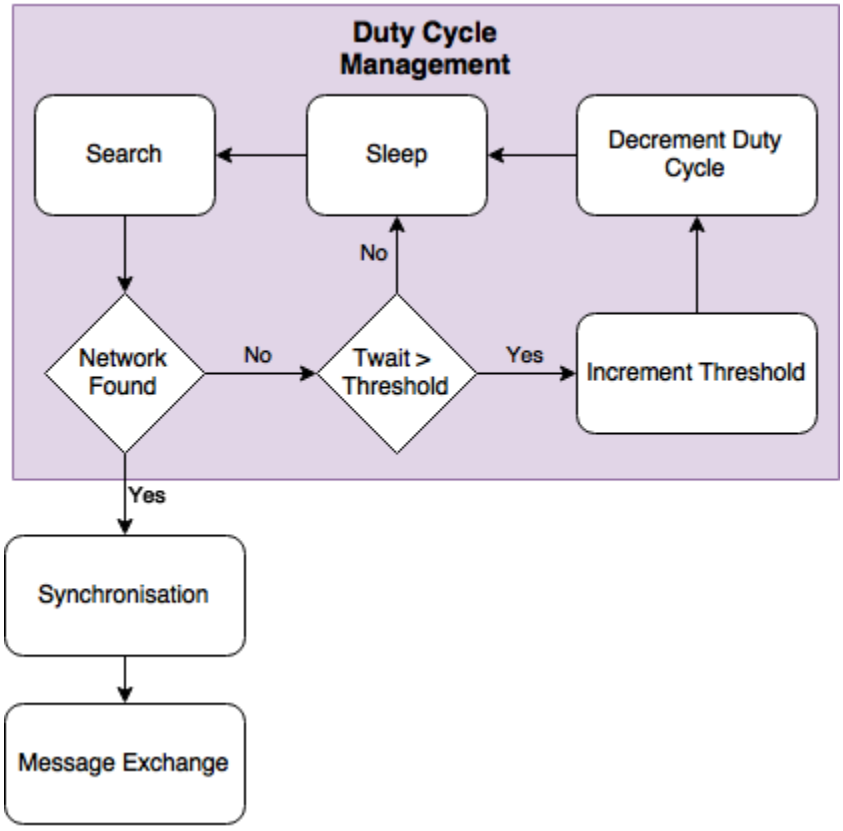


Figure 25: Mote Join Routine Logic

5.2.9.1 Duty Cycle Management

Initially the mote will begin looking for a network using a predefined search function [4]. If the mote finds a network, it will move on to a synchronisation state. If not, the mote will sleep for a period of time before starting the

search again. Cumulative searching time can range from ten seconds to tens of minutes, depending on the join duty cycle [20]/ This is the proportion of time a mote spends listening for a network versus sleeping. This is able to be configured by the user in the range 0-255, with 255 representing a 100% duty cycle. This is implemented with an autonomous function that the user has little control of other than increasing mote density by location or turning the setting off explicitly at the manager.

It is important to note the mote will have a higher average current if a larger duty cycle is specified, however it will also find the network quicker [6]. There is a trade-off that exists between the speed when a network is present and how much energy is used when it isn't. To help manage this, CSC has designed the following duty cycle management process as seen in Figure 25:

- An initial threshold is set at thirty seconds, with a duty cycle of 100%.
- If a network has not been found after thirty seconds, the threshold is incremented to two minutes, and the duty cycle is decreased to 80%.
- If no network has been found after two minutes, the threshold is incremented to ten minutes, and the duty cycle is decreased to 50%.

The process will continue to decrease the duty cycle over time, thereby preserving battery life if there is no network to be found. The values used in the example above are for reference only; these will be customised to the user's application.

Implementation of this duty cycle management process involves modifying the QSL libraries to expose the duinoPRO API layer underneath. Due to time constraints, CSC was unable to develop a working prototype of this subroutine.

5.2.9.2 Synchronisation

Once the mote has found the network, it enters a synchronisation stage where it sets its internal parameters to match the network. The mote is only in the synchronisation state for a few seconds, and there are little to no configurable parameters during this stage [6].

5.2.9.3 Message Exchange

Once the mote has synchronised to the network it is able to receive and send messages to other motes. A typical mote-to-mote exchange takes approximately ten seconds, but can increase significantly based on two factors:

- Downstream bandwidth: how much data is able to be sent outwards from one mote [6].
- Number of motes: contention among many motes simultaneously trying to join compete for limited resources, slowing down the joining process due to collisions between exchanges [6]

A larger duty cycle will enable one mote to join a network quickly, however it will also result in a large number of collisions if a large number of motes are in use. For a large number of motes, a lower duty cycle will result in faster joining time and less power consumption [6]. This was another reason CSC designed the duty cycle management process outlined in Section 5.2.9.1.

5.2.10 Sample – Payload Formation

The process of sampling data and forming an appropriate packet payload for transmission through the WSN has been abstracted to a single function, denoted as "sample". The process of sampling data and formatting it into packet payloads is critical to achieving the functionality of the system in providing an unbroken data flow from sensor to cloud database.

The "sample" function implemented for this task is called from the main mote routine (Section 5.2.8) when it is determined that a sample should be procured. Construction of the entire payload, as per the conventions outlined in Section 3.2, is managed by this function. Additionally, this function takes a single argument which allows the mote main routine to specify which sampling mode to perform. The available options are listed in Table 16.

Table 16: Sampling mode options

Function argument	Sampling mode
1	Sensor data only
2	Diagnostic data (battery voltage) only
3	Both sensor and diagnostic data

At the completion of its execution, the 'sample' function returns success or failure. On success, the full payload is written to C uint8 array in global scope, ready for transmission. On failure this variable is cleared.

The implementation of this process utilises a layered approach, with the 'sample' function as the top layer. This is illustrated in Figure 26. Function names are written in italics.

As shown, the 'sample' function sets the payload headers and then calls the appropriate sample functions in the next layer given the sampling mode specified (Table 16). These data sampling functions manage data collection and write their own field headers and values (data) to the payload directly. Data collection is mediated via the data acquisition functions in the next layer, while writing the field header and reserving space for collected data is mediated by the framing functions in that layer. Finally, low-level drivers are called by the data acquisition function for direct hardware interfaces. This layer is largely outside the scope of this project, as discussed in Sections 5.2.11.2 and 5.2.11.3.

The design of this layered approach was developed with consideration of requirement M03. In particular, each layer was allocated a well-defined scope and functions with similar purposes were implemented with similar structures as far as practicable. This approach aids in development of a clear and maintainable code base (requirement M03).

The remainder of this section describes the implementation of the 'sample' function, data sampling functions and framing functions. Data acquisition functions are described in Section 5.2.11.

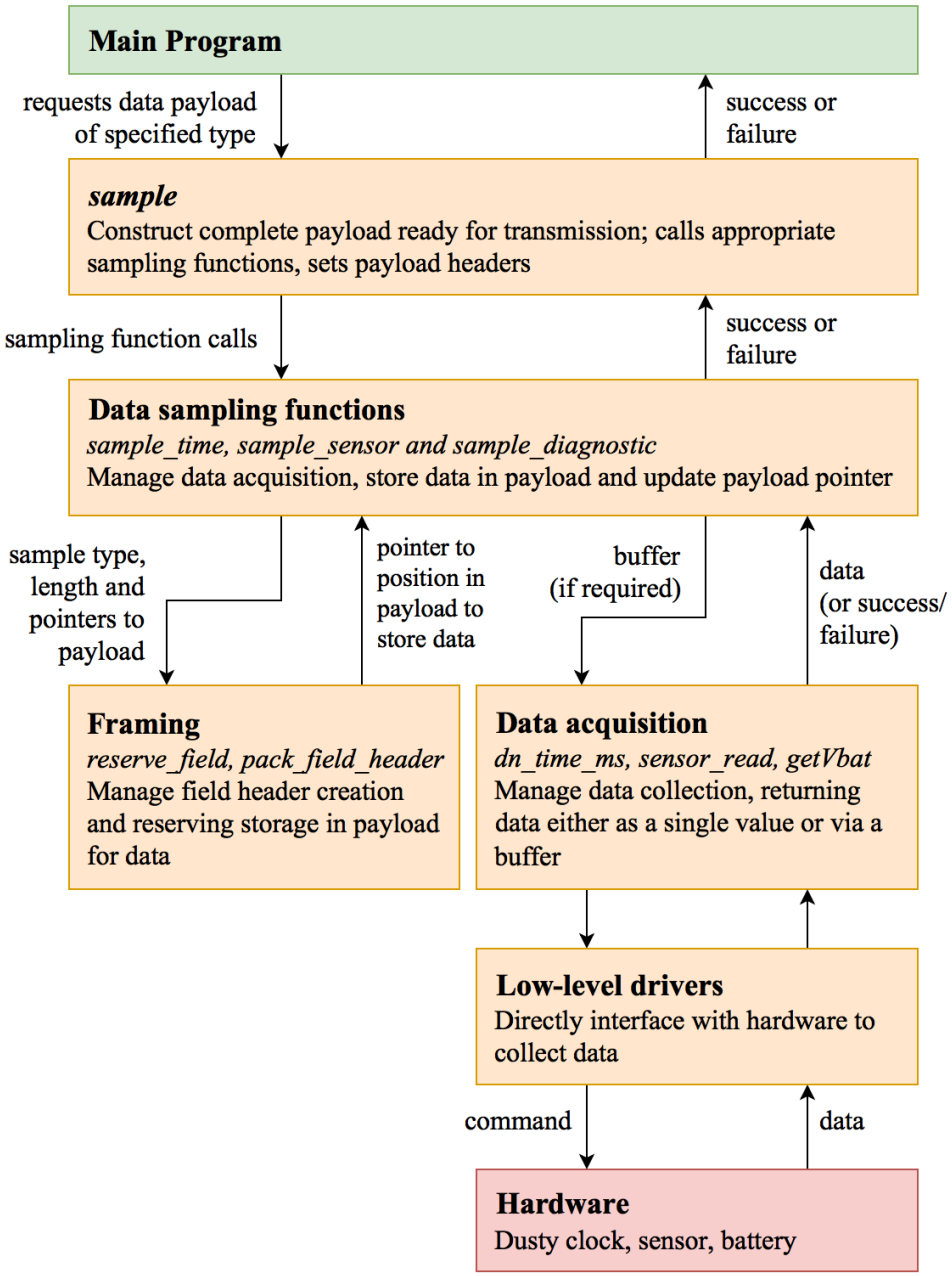


Figure 26: Layered approach to sampling

5.2.10.1 Sample function

As described above, the ‘sample’ function is responsible for managing data sampling and payload construction. This function takes a single argument that specifies the type of sampling to undertake (Table 16). Its primary tasks are to set the payload and dataload headers and call the appropriate sampling functions. Figure 27 illustrates the design.

As shown, the argument is first validated and the payload flushed. Next the payload header is set to 0, indicating that it is of type ‘data’ (see table with options, config, data?). The current network timestamp is then stored in the payload using sample_time(). Next, the sample_sensor and/or sample_diagnostic functions are called to collect the relevant data and write this to the payload. A check is then implemented to ensure the payload does not exceed its allowed length. Finally, the payload is marked as ready to send by setting its _ready_sent parameter to 1.

Integration tests of this function and its interaction with the data sampling functions have been completed successfully as outlined in the testing document under test TU_SH/DpSample_SampleMain_Op (ref).

The checks mentioned herein are employed to ensure only valid data is transmitted, pursuant of requirement M07.

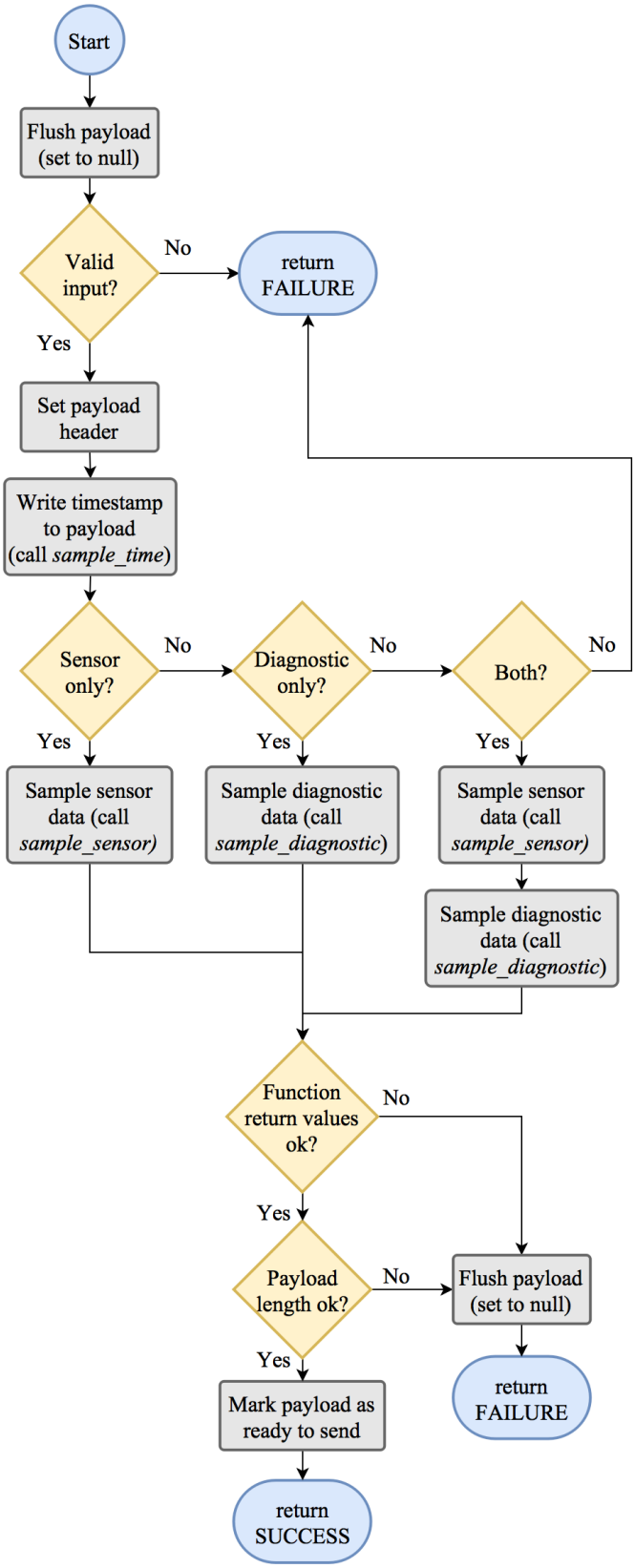


Figure 27: Sample function flowchart

5.2.10.2 Data sampling

The data sampling layer sits immediately below sample and consists of three functions. Three functions are provided in this layer for sampling different data, as listed in the first two columns of Table 17.

Table 17: Data sampling functions

Function	Data sampled	Data acquisition mechanism
sample_time	Dust network time	Send getParameter<time> command to Dusty module [21]
sample_sensor	Sensor reading(s)	sensor_read function (Section 5.2.11.3), which uses sensor-specific duinoPRO libraries provided by ATAMO (with some modification)
sample_diagnostic	duinoPRO battery voltage	duinoPRO board enableVbatSense and getVbat methods [22]

Though each function samples different data, their structures are identical. A generic template of this is illustrated in Figure 28. As shown, reserve_field is first called to reserve space in the payload for the data that will be collected and write the correct field header to the payload. Second, data is acquired using the appropriate mechanism as listed in column three of Table 17 and described in Section 5.2.11. Third, the data is checked for validity (requirement M07). Finally, the data is written to the payload, in global scope. Full documentation for these functions is available at (ref).

Unit tests of all three sampling functions have been completed successfully as outlined in the testing document under tests TU_SH/DpSample_SampleTime_Op, TU_SH/DpSample_SampleTime_Op and TU_SH/DpSample_SampleSens_Op (ref).

Requirement M08 is addressed by providing for 16-bit sensor resolution. However, it is the sensor driver function that is responsible for procuring the data in the correct (up to 16-bit) format (Section 5.2.11.3).

Requirement A01 is met and exceeded here by providing functionality to obtain diagnostic data and incorporate it into the payload (sample_diagnostic function). 32-bit precision is provided, exceeding the minimum limit of 16-bits set in requirement A01.

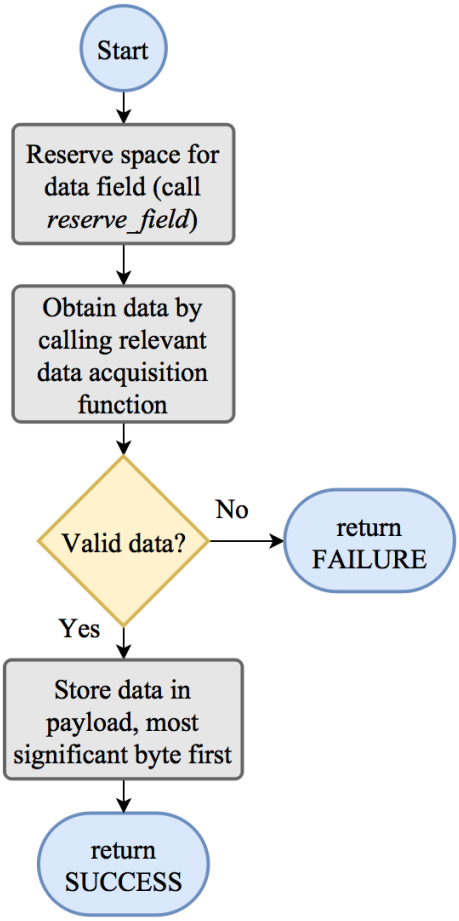


Figure 28: Data sampling functions general structure

5.2.10.3 Framing

The framing layer fulfils the role of reserving intervals within the payload for data to be written (ensuring conflict-free writing) and forming and writing field headers at the beginning of the reserved intervals (ensuring data can be interpreted downstream). Two functions are provided for this task as listed in Table 18 and illustrated in Figure 29 and Figure 28Figure 30.

Table 18: Framing layer functions

Function	Process	Return value
reserve_field	Calls pack_field_header, then increments the payload pointer to reserve the required amount space	Pointer to the start of the field value in the payload
pack_field_header	Generates the correct field header and writes this to the payload	Length of field header

The *payload pointer* is a variable that tracks the next available position in the payload (which is a C uint8 array) that has not yet been written to. As such, reserve_field reserves space in the payload array by incrementing this pointer by the appropriate number of bytes. However, it also returns a pointer that points within that (reserved) interval to the first entry where new data can be written. This return pointer is then used in data sampling functions for writing data to the payload.

The pack_field_header function generates the correct field header given the field value length (in bytes) and type (timestamp, sensor, diagnostic) based on certain rules. First, the field value length is encoded into the first three bytes of the header by dividing it by two and storing it in the three MSBs of the field header as shown in Table 19. At present, even-numbered lengths of 2-12 bytes are supported. However, this only requires six of the eight options available, leaving room for expansion. Second, the type of field is encoded into the five LSBs of the field header according the rules listed in Table 20. This set of rules allows data packets to be easily interpreted by downstream applications, namely the network manager.

Table 19: Field header length indication

Field length (bytes)	Field header
2	0b001XXXXX
4	0b010XXXXX
6	0b011XXXXX
8	0b100XXXXX
10	0b101XXXXX
12	0b110XXXXX
Other	Invalid

Table 20: Field header type indication

Field type	Field header
Timestamp	0bXXX11111
Diagnostic (battery voltage)	0bXXX11110
Sensor reading	Sensor type code (SECTION)

Note that neither function in the framing layer directly accesses the payload variable. This was a design decision to limit access rights to only the data sampling layer and the push function (SECTION).

Unit tests of both framing functions have been completed successfully as outlined in the testing document under tests TU_SH/DpFrameHdr_PayloadHdr_Op and TU_SH/DpReserveField_Payload_Op (ref).

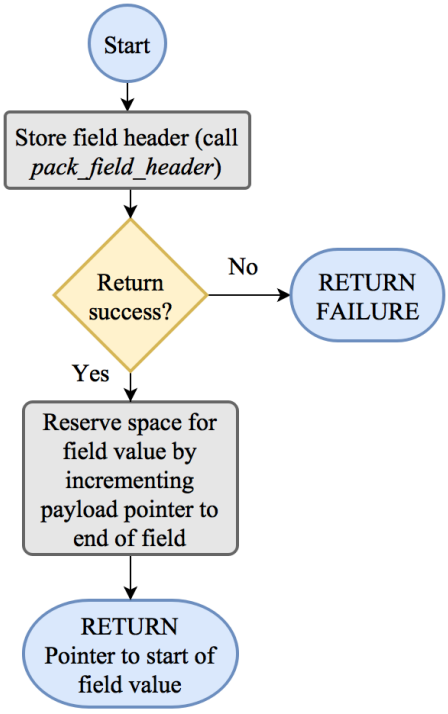


Figure 29: reserve_field function flowchart

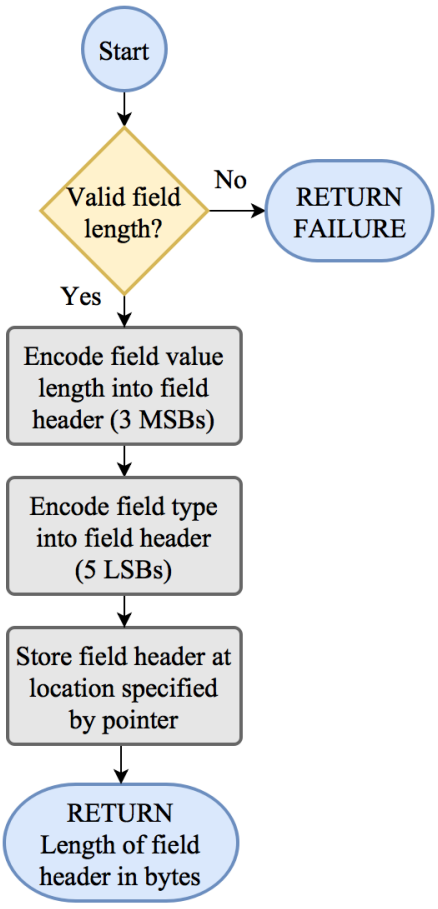


Figure 30: pack_field_header function flowchart

5.2.11 Data Acquisition

Three data acquisition methods are required in the layered approach to sampling illustrated in Figure 26. This section explains the methodology employed for each of these. Note, however, that the low-level driver tasks are considered outside the scope of this project as explained in the respective sections below.

5.2.11.1 Timestamp

The Dusty module network timestamp is required as the first field in any payload sent by a mote. Obtaining this time from the Dusty module is possible through a number of means. First, the TIMEn pin on the Dusty module may be strobed, upon which the Dusty will send a time notification packet to the duinoPRO [21, p. 82]. This is most accurate; however, the chosen method is to send the getParameter command to the Dusty module to request the current time from it [21, p. 47]. The motivation for this is to free up all six IO pins on the duinoPRO for UART Mode 2 communication with the Dusty module (see Section), which is more robust with regards to timing and more energy-efficient [18].

Implementation of this functionality is via the QSL, which must be coded for the specific hardware used in this project [23]. This has not yet been implemented.

5.2.11.2 duinoPRO Battery

Measurement of battery voltage on the duinoPRO is facilitated by duinoPRO firmware provided by ATAMO. Specifically, the battery voltage can be queried using the getVbat member function of the duinoPRO class [22]. The voltage is returned as a 32-bit floating point number. Note this is later subdivided at the bit level into four uint8 values for storing in the packet payload, as described in SECTION.

Before reading the duinoPRO battery voltage, the voltage sensing mechanism must be enabled [22]. This should then be disabled once the battery voltage has been obtained so that the voltage sensing module does not consume power needlessly.

The entire process of enabling battery voltage sensing, procuring a value and disabling sensing is managed within the sample_diagnostic function discussed in Section 5.2.10.2. Note that the actual functions for reading and enabling the battery voltage are considered “low-level drivers” (as per Section 5.2.10) and are therefore outside the scope of the project. These functions will be treated using a ‘black box’ approach.

5.2.11.3 Sensor

In a similar way to how sampling has been abstracted and implemented using a layered approach (Section 5.2.10), reading from the sensor has likewise been abstracted to a single function called “sensor_read” with multiple software layers below it. The implementation of this approach is illustrated in Figure 31. As shown, sensor_read is called by the data sampling function sample_sensor (Section 5.2.10.2). It is responsible for ensuring a sensor has been configured and then selecting the appropriate sensor driver to run. The next layer consists of the sensor drivers themselves. Below this is a layer which contains the sensor libraries, which are developed by ATAMO for interfacing with specific sensors supported by the duinoPRO.

Note that the sensor-specific libraries are considered to be “low-level drivers” (as per Section 5.2.10) and are therefore outside the scope of the project. The sensor driver layer will call functions from the sensor libraries; however, modification and/or development of these libraries if the domain of ATAMO and outside the scope of this project.

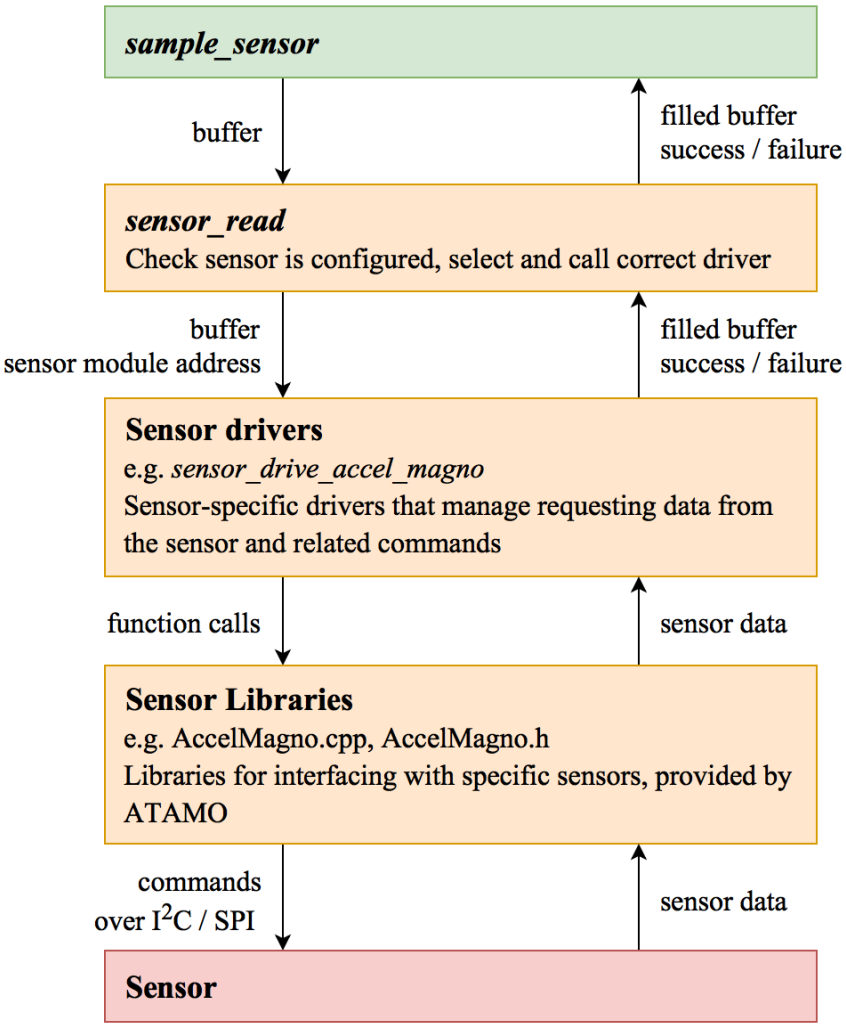


Figure 31: Layered approach to sensor data reading

The structure of the sensor_read function is illustrated in sensor_read function. As shown, it first checks that sensor configuration has been set. This prevents sensor_drive from running without a configured sensor, which could cause an error. Next, the appropriate sensor driver is selected and called based on the type of sensor connected to the suinoPRO. The same buffer passed to sensor_read is then passed on to the sensor driver. Finally, the return value of the driver is checked. The buffer is reset to zero and a failure state recorded if it returned unsuccessfully (addressing requirement M07).

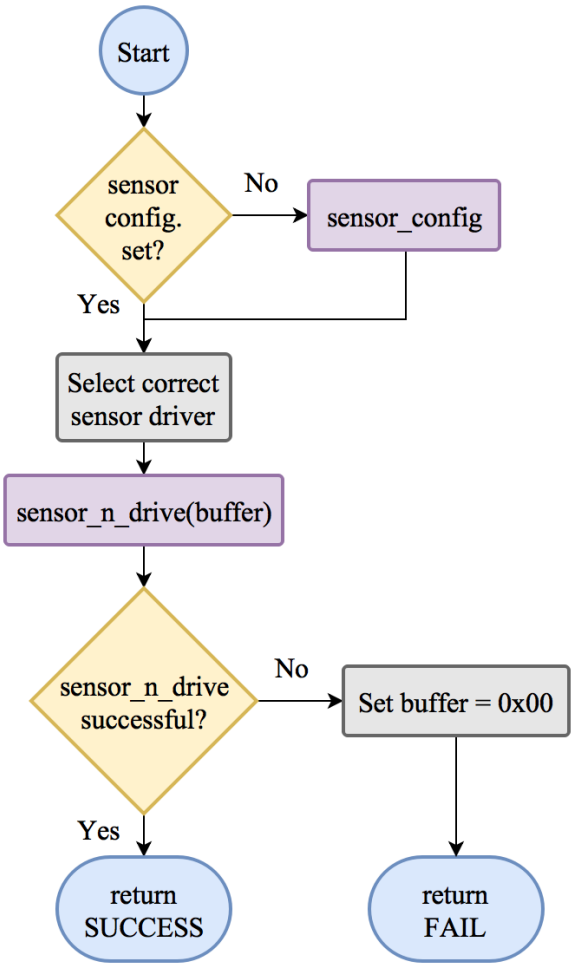


Figure 32: sensor_read function

Sensor driver implementations are likely to vary significantly between sensors. However, as described above, the general philosophy is to use ATAMO-developed sensor-specific libraries for the low-level tasks and simply call high-level library functions for tasks such as: enabling sensor outputs, changing sensor mode and requesting sensor readings.

One specific case has been implement for the LSM303D accelerometer/magnetometer sensor [24], which was provided to CSC by ATAMO. Libraries were also provided for this sensor; however, they required some modifications to be compatible with the current duinoPRO firmware [25]. The sensor driver function implemented for this sensor is structured as follows:

1. Initiate communication with sensor
2. Exit sensor low power mode
3. Enable outputs and set output scales
4. Request data and write to buffer
5. Return sensor to low power mode

Of note is the use of the LSM303D's low power mode, in which it consumes 1/300 of its normal operating power [24]. The sensor will be in its low power mode at all times except while a sensor reading is being obtained.

5.2.12 Sensor Configuration

To address requirement M11, CSC have designed a routine called sensor_config that automatically scans the module spaces on a duinoPRO sensor-host for supported sensors and records pertinent configuration data in global state variables. This approach greatly improves the portability of the system since an end user can attach any supported sensor to any module site on a duinoPRO sensor-host and still use the same software on the duinoPRO.

The structure of this subroutine is illustrated in Figure 33. As shown, the default sensor configuration is checked first. That is, the default sensor address is polled using the default serial communications mode. If an 'expected response' is received, this is taken to mean that the default sensor is connected. The subroutine then records that a sensor is configured (_sensor_conf_flag set to 1) and returns success, leaving the default values untouched.

If the expected (default) response is not received, the subroutine loops through all available module sites using both I^2C and SPI, polling each combination in turn. Responses are checked against a lookup table (LUT). This table contains a list of 'valid responses' (to polling) and associated 'type codes' that uniquely identify the type of sensor connected. These type codes are used by sensor_read to determine which sensor driver to call. If a valid response is found, that address and type code are stored in global state variables *_sensor_addr* and *_sensor_type*, respectively. If no supported sensor is found, sensor_config returns failure.

The default sensor configuration is an LSM303D accelerometer/magnetometer sensor [24] connected via SPI [26] at module address three. Module three was selected by CSC since it is at a corner of the duinoPRO (accessible) and far from the Dusty module (minimal EMI).

A critical impediment to this design is the limited memory of the duinoPRO (constraint C01). That is, there may be insufficient memory to store a large number of sensor drivers on the duinoPRO simultaneously. An alternative approach that would reduce the portability of the system (requirement M11) but avoid this constraint is to include only one sensor driver in the duinoPRO build at compile time. With this approach, the user would potentially have to install different software on each sensor-host in order to meet requirement M11 and support multiple sensor types.

At present, CSC has only implemented this alternative approach due to time constraints.

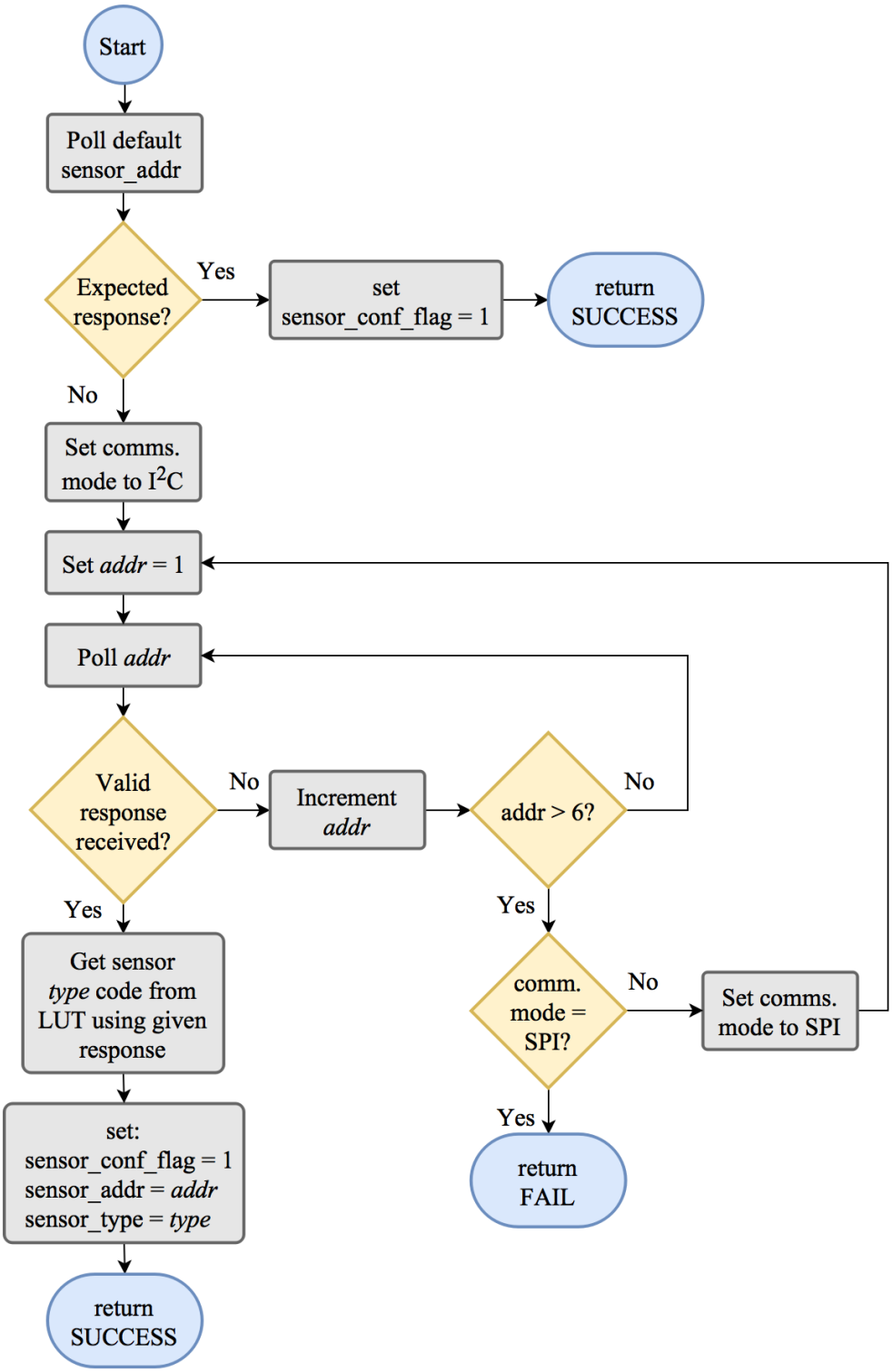


Figure 33: sensor_config subroutine

5.2.13 Mote Configuration

A set of functions have been designed for managing mote configuration. These are responsible for setting default configuration parameters when required and updating these whenever configuration packets are received from the network manager. Configuration data is stored in C array in global scope (_conf). Four functions are included in this design; their interactions with each other, the mote main routing and the configuration data array are illustrated in Figure 34.

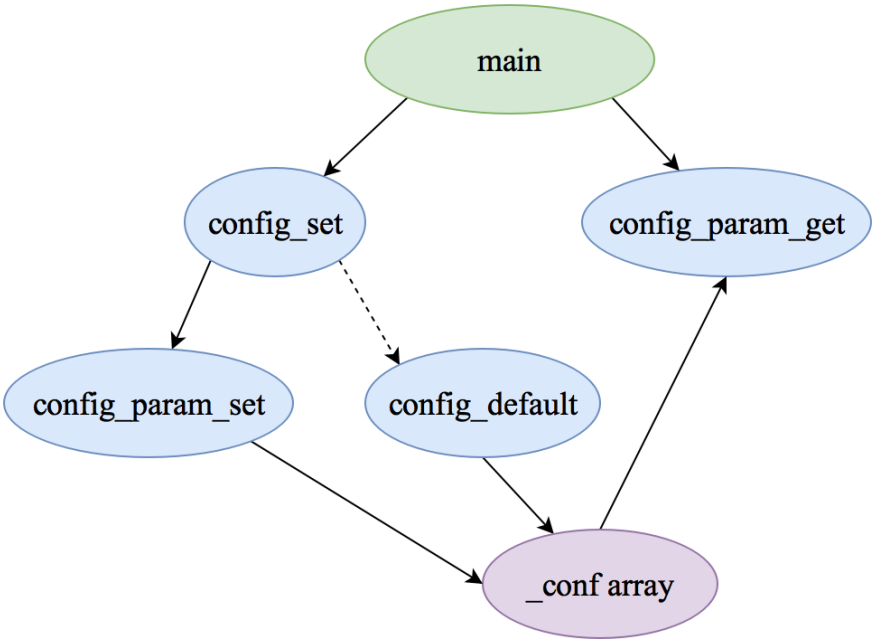


Figure 34: Configuration functions interactions

5.2.13.1 Configuration State Variables

Sensor mote configuration requires two state variables:

1) *_config_set_flag*: 0 by default (not configured), changed to 1 when config_default has successfully initialised *_config*.
2) *_conf*: character array of configuration parameters. The list of configuration parameters is documented at [27] and is expected to grow as the system design develops.

5.2.13.2 Configuration Lookup Table

The configuration lookup table is required for several configuration functions. It contains five columns of data for each configuration parameter:

1) Field value length (number of bytes)
2) Maximum parameter value
3) Minimum parameter value

Note that the field header value for a given configuration parameter will be identical to its row number in the lookup table and its index in the _conf array. Hence, the lookup table need only include the three values listed above.

5.2.13.3 Configuration Set

The config_set function ensures configuration parameters are set, loading default values if required and parsing a configuration packet if supplied. It will be called at start-up to initialise configuration parameters and whenever a configuration packet is received from the network manager. Overall, this achieves requirement A03. It takes two arguments:

1) A configuration payload sent from the network manager, or else a dummy variable, and
2) The length of the payload in bytes, or zero if the dummy variable was passed.

This function first checks its arguments and device state. If the dummy variable and zero are provided as arguments or *_config_set_flag* = 0, config_default is called (see Section 5.2.13.5). If length is greater than zero, config_set parses the configuration payload, updating each configuration parameter contained therein. It is assumed that the calling function has already determined that the payload is of the correct type (i.e. configuration).

However, if any field header is invalid (not found in lookup table (LUT)), then config_set returns failure irrespective of how much of the payload it has parsed. This is necessary because the function would not know how many bytes to 'skip' for the Field Value of the invalid field and therefore could not proceed.

At termination, the function returns the number of parameters in the payload it was unable to update.

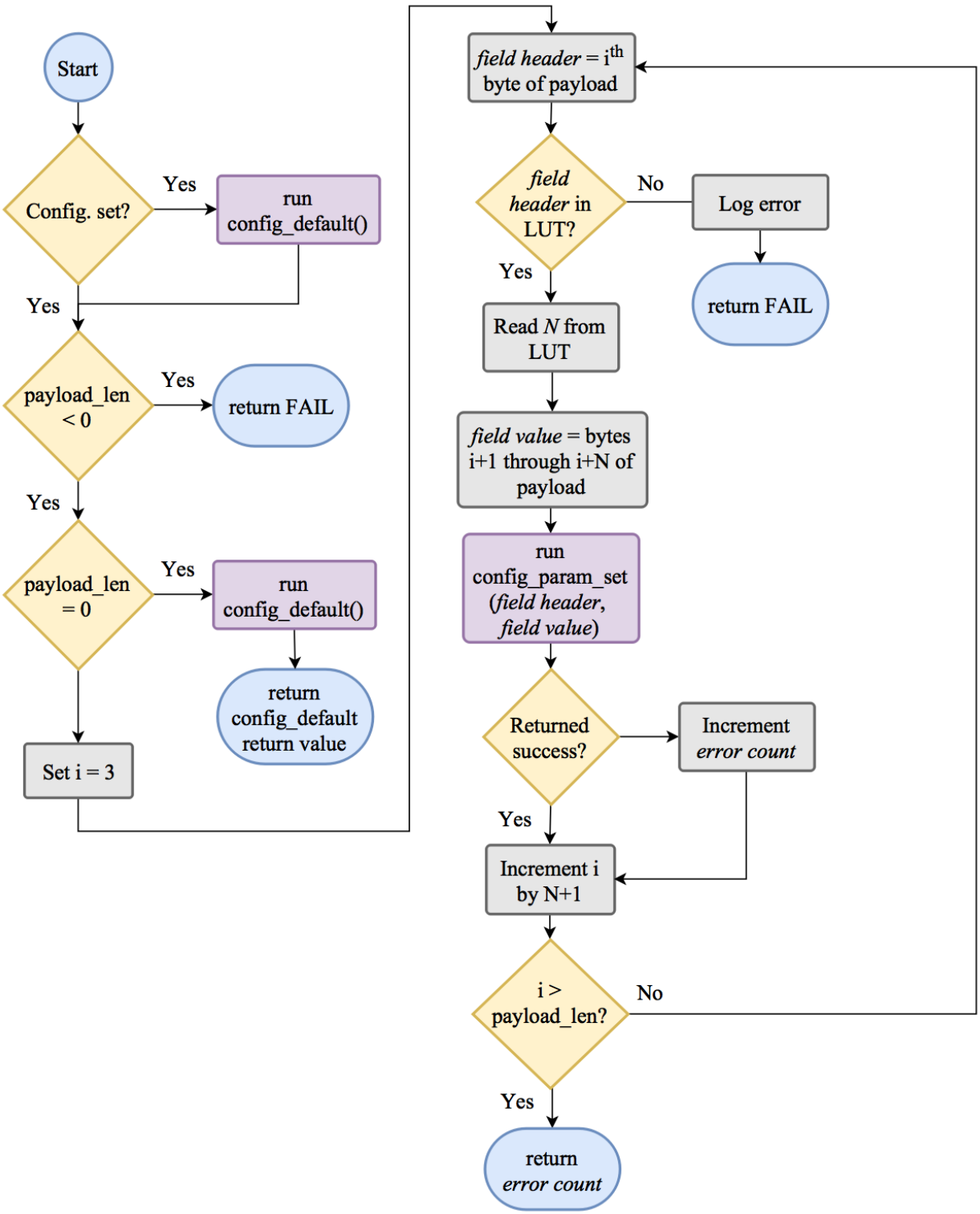


Figure 35: config_set function

5.2.13.4 Configuration Parameter Set

The config_param_set function updates an individual configuration parameter. It takes two arguments:

1) Field Header (*type*)
2) Field Value (*value*)

Before assignment, the provided parameter value is checked against its minimum and maximum values in the configuration lookup table. The function is illustrated in Figure 36.

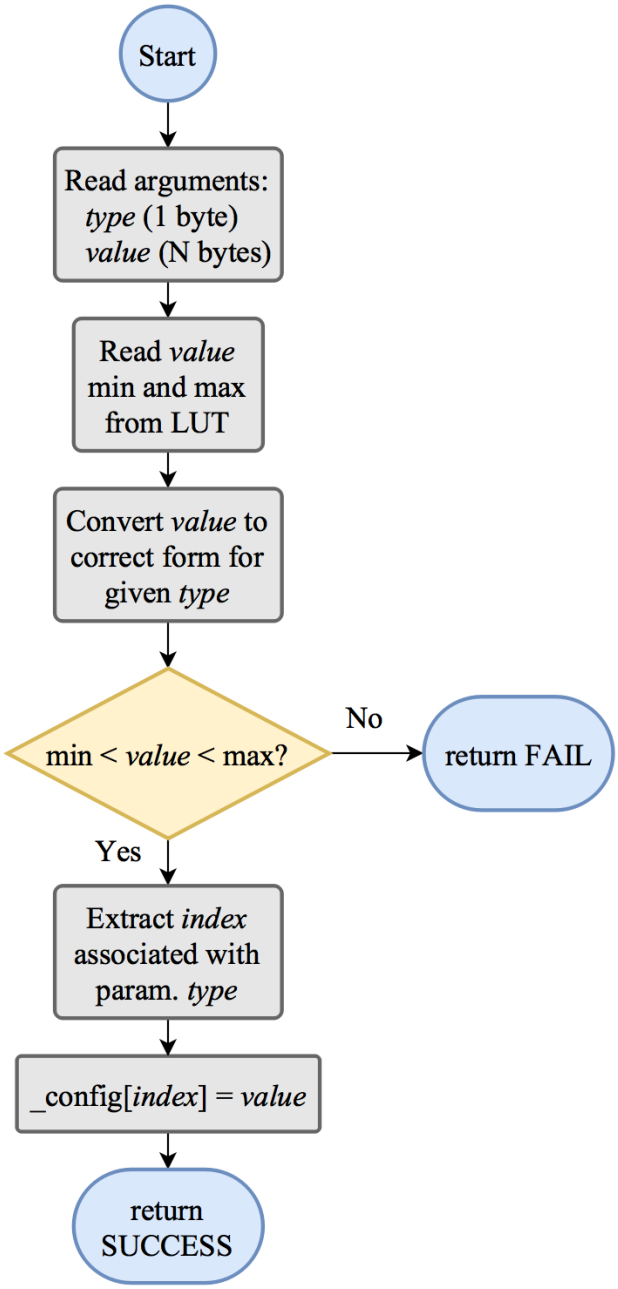


Figure 36: config_param_set function

5.2.13.5 Other Configuration Functions

In addition to the above, two other configuration functions are required:

1) config_default: this function takes no arguments and sets configuration parameter values to default values that are hard-coded into the program.
2) config_param_get_cur: this function takes a field header as its argument and returns the current value of the associates parameter.

5.2.14 Network Manager and Gateway

This section discusses the key design decision made in the design of the network manager and gateway device.

5.2.14.1 Embedded Network Manager

In a SmartMesh IP network there are two choices for the Network Manager. They are the Embedded Network Manager (EManager) and the Virtual manager (VManager) [20]. The Embedded Manager combines the network management functions and access point onto a single chip [20]. The virtual manager communicates with many separate access point motes via an x86 server [20]. The virtual manager has the advantage of supporting thousands of motes where the embedded manager can support up to 100. The VManager is significantly more complicated to set up. Requirement A02 sets a target of only 5 motes to connect to the network manager. Due to the associated time constraints, the implementation of VManager (Requirement A09) was given low priority by the client the embedded manager was chosen to develop the first prototype.
Initially the DC2274A-A Network Manager dongle will be used however in the future a dusty module may be flashed with the EManager Firmware and used instead [28]. Currently communication have been established between the DC2274A-A and a laptop.

5.2.14.2 Gateway Application

The gateway application will run off a separate device which communicates with the manager over the serial application programming interface (API) [29]. The API functions using commands and notifications. Commands are requests sent from the gateway device to the manger, and notifications are messages from the manager to the gateway containing useful information from the SmartMesh IP network [29]. The application will be developed in Python 2.7 using and the SmartMesh SDK which fully implements the API in Python [30]. The SDK example projects will also aid to speed up development time. Currently the application is being developed for a laptop but any device running the SmartMesh SDK in Python will can run the application.
The main functions of the application are to:

- Process data from the network and upload it to the cloud to meet M06
- Update mote Configuration as instructed from the user to meet A03
- Log key network events that are sent from the manager

A flow diagram of the gateway application logic is shown in Figure 37.

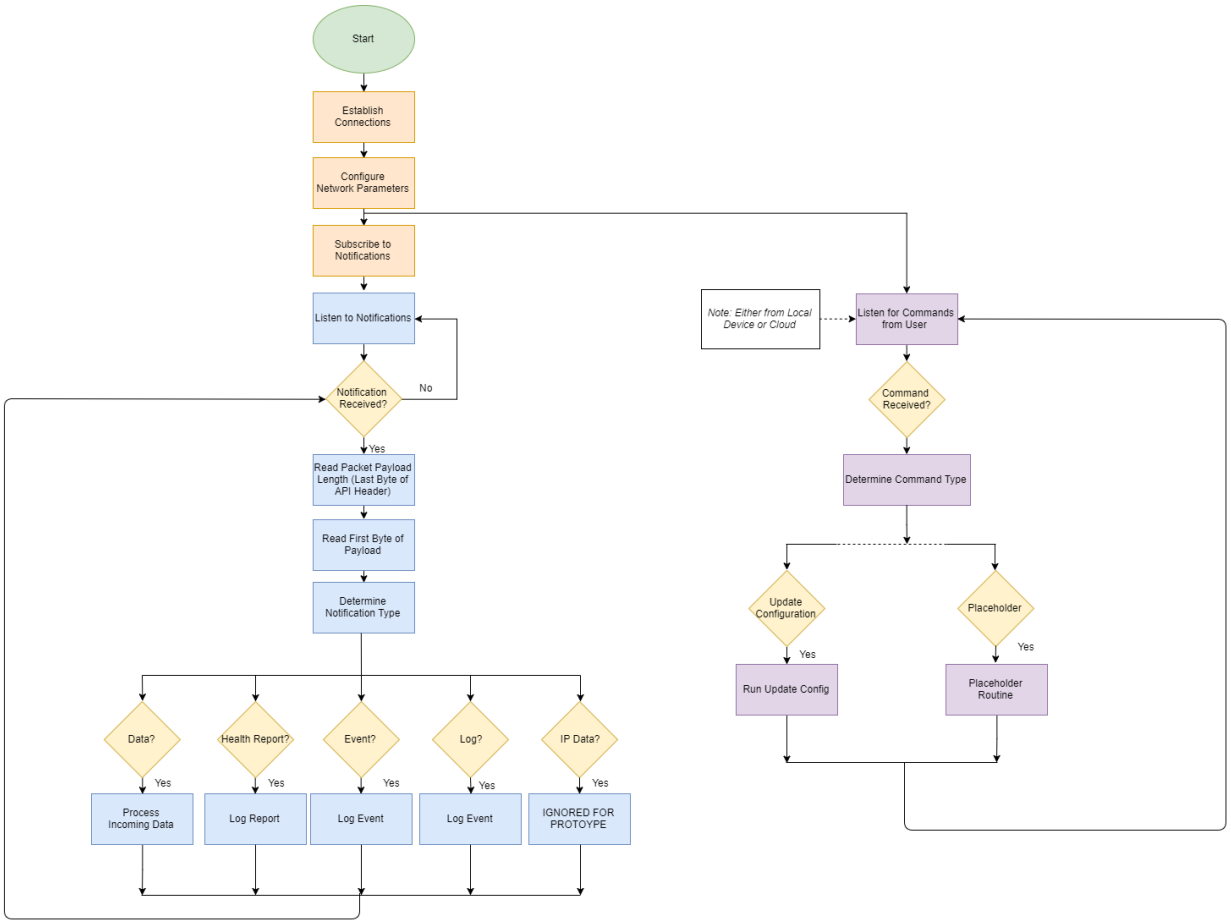


Figure 37: Gateway Logic Diagram

Firstly the application will establish a connection with the network manager and the cloud server. Next it will send the *"subscribe"* command over the API to control which notifications it must acknowledge [29]. If any of the commands that were subscribed are not acknowledged the connection will be restarted.
Next the application will listen to the API for notifications from the manager and listen to the cloud server and/or the command line interface (CLI) for commands from the user. When a notification is received the system will first read the packet payload length from the API Header. It will then read the first byte of the payload which indicates the command type. This will then control which routine is run to correctly process the data. Next it will loop back and listen to the API for another notification. In parallel the application will listen for commands from the user. The logic for this process follows an analogous process.

Currently the commands from the user were delayed due to time constraints. The connection between the laptop and the embedded manager has been established and it was confirmed that the application has easy access to the payload data and mac address. This confirms that the method of process incoming data described in the next section will work. The function to process incoming data is partly coded but is yet to be tested as this stage of the design the motes are yet to send any data to the network manager. The designers of the front end have developed and tested a python function to upload the processed data to the cloud database.

5.2.14.3 Upload Mote Data Routine

This function takes the incoming data notification and unpacks the data before sending it to the cloud through the boto3 python package. A diagram of the logic for this process can be found in Figure 38.

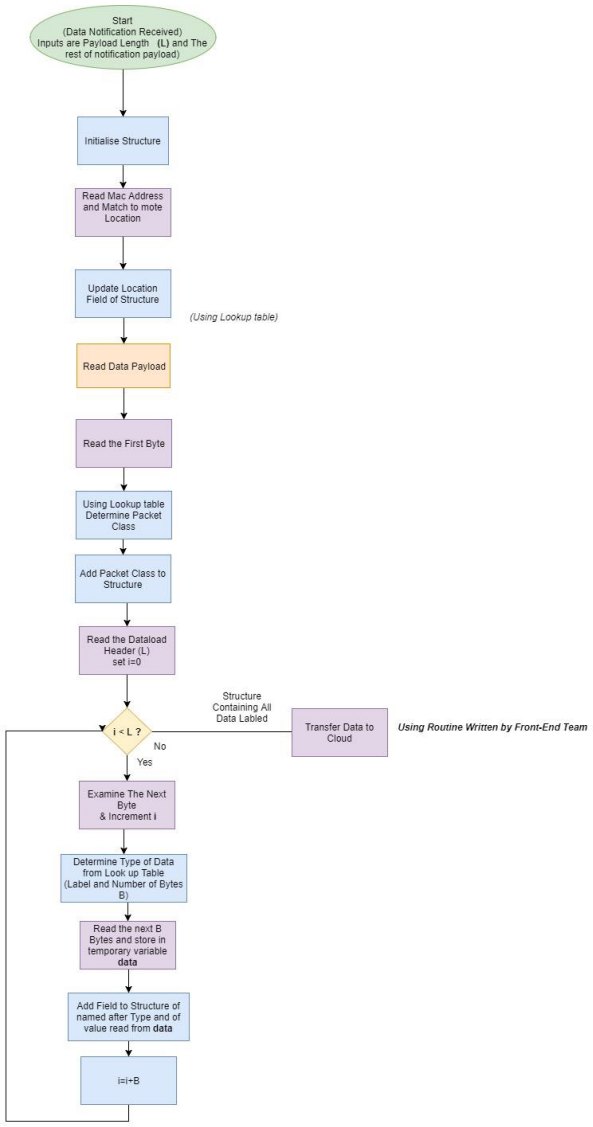


Figure 38: Process and Upload Mote Data

The programs first step is to calculate the number of bytes of data in the payload. This is found by subtracting the amounts of bytes taken up by notification type (1), timestamp (12), MAC address (8), source port (2), and destination port (2) which are placed at the front of a notification payload from the payload length (L) found in the API header [29]. Then the program creates a blank dictionary that data from is stored into. The program ignores the first byte as the notification type is already known, and then ignores the next 12 bytes. The MAC address is then read from the next 8 bytes to identify the mote which sent the data. This location is added as a field to the dictionary. The next four bytes are skipped.

Following this the program runs through a loop which extracts data and adds it to the dictionary. Each loop first examines one byte to determine the type of data it is reading. This type provides a label for the data and the length of the data in bytes (B). The program then reads the next B bytes and uses the label to populate a new field in the dictionary. At the end of each loop the variable **i** is increased by (B+1). The loop will end when **i** reaches N indicating that then end of the data has been reached.

The application is then left with a Python Dictionary which it must upload to the cloud. The cloud sub-team have

stated that the Node-Red application will be implemented to manage the flow of data into their sub-system. This will be achieved using a free MQTT broker which can be accessed by Node-Red [31]. This approach is based on an example in the SmartMesh SDK [32].

5.2.14.4 Update Configuration Routine

The routine is called if the user sends a command to the gateway stating a specific change to one (or many) configuration parameters of the motes. A flow diagram demonstrating the logic of this process is shown in Figure 39. Firstly, the program calculates the number of parameters that need to be updated. It then sets the first byte of the packet payload to indicate that it is a configuration packet. This indication is used by receiving motes to initiate their update configuration sub-routine. The program then goes through a loop until all parameters are added to the packet. Finally, the configuration data is sent out to all motes using the *"send data"* command over the API. The packet is sent to all motes by setting the address to 0xFFFFFFFFFFFFFFFF when calling the "send data" data command [29]. At this stage this routine is only a plan for how the team would do this task. The team has not yet had the opportunity to implement or test it.

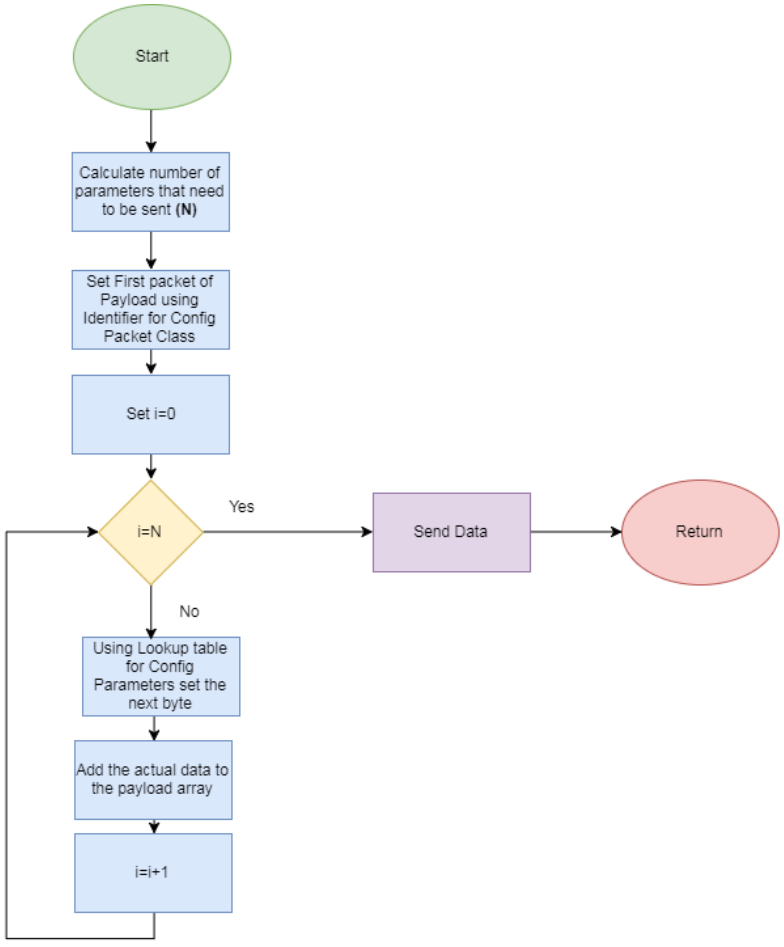


Figure 39: Update Configuration Routine

5.3 Front-End Design

In achieving elevated data visualisations and remote monitorisation, CSC has integrated cloud services into the WSN design illustrated by the flow diagram in Figure 40. The cloud database serves as a convenient remote storage system which interfaces between the embedded manager and the web application [9, 33]. The successful integration of the cloud database is contingent on its ability to conveniently, flexibly and reliably store and query data. From the requirements analysis it was determined that the embedded manager required the cloud database to store timestamps, unique mote identification numbers, sensor data and diagnostic data [34]. The design of the cloud integration and web application are deeply intertwined and are aimed to optimise the quality benchmarks of the WSN whilst meeting the prioritised requirements.

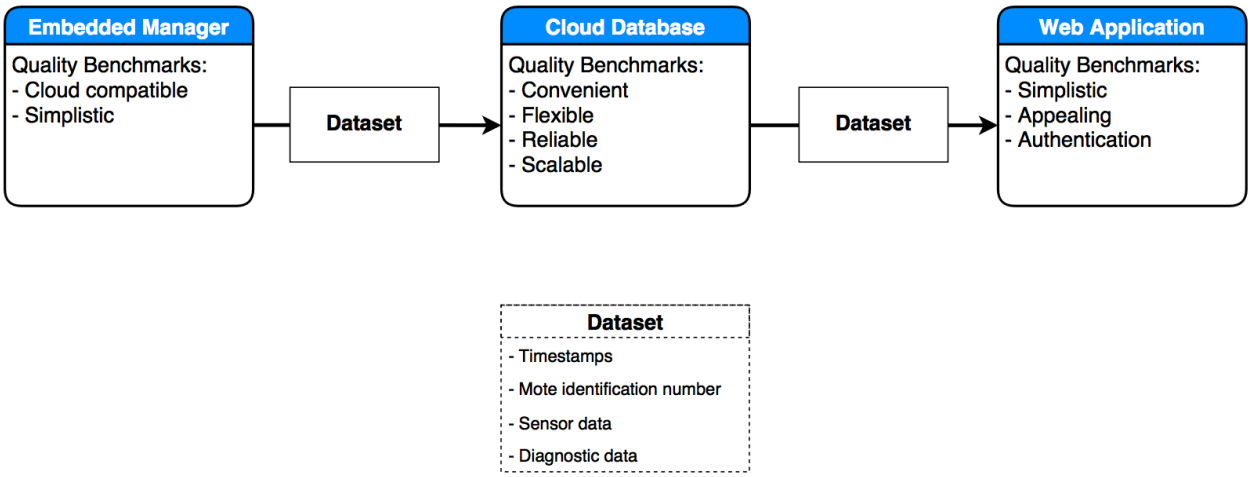


Figure 40: Front-End Intrasystem Integration

5.3.1 Cloud Integration

Upon the successful data warehousing from the WSN, the network manager will store its datasets into a NoSQL cloud database. CSC has chosen AWS as the cloud service provider due to its elevated support in developing basic applications when compared to International Business Machines (IBM) Watson. In utilising the AWS resources, CSC has chosen Amazon DynamoDB for its ease of deployment, schema-less characteristics and horizontal scaling.

5.3.2 Choice of Cloud Service Provider

To select a suitable Cloud Service Provider (CSP), CSC conducted an analysis on the client's preferred solutions – IBM Watson and AWS [9, 33]. In analysing the cloud-integration processes, it was concluded that IBM Watson and AWS had the capacity to effectively interface with the embedded manager. Enabling the WSN to store information into the cloud database near real-time. Thus, partially satisfying the relevant mandatory requirements dictating compulsory internet/database connectivity (Requirement M06) and data timeliness (Requirement M09). The comprehensive engineering support provided by IBM and AWS also bolster the design strategy facilitating for the concise documentation requirement (M03) [35, 36].

In optimising the system costs, it was determined that CSP expenditures were heavily correlated to the solutions architecture. Evidently, CSC asserted that minimalistic cloud solutions would minimise system costs (Requirement A07) as superfluous functionalities would result in additional expenditures. Therefore, the main determinant in the choice of CSP was ultimately based upon their perceived level of necessity and synergies to the WSN [35, 36].

IBM Watson is a cloud computing platform specifically designed to replicate cognitive abilities using machine learning [36]. It develops artificial intelligence by storing information into a managed NoSQL JSON database [36,

37]. Although the IBM Watson services are beneficial in general IoT environments [36], the additional benefits are non-essential as it does not improve the client's benchmarks of quality. Therefore, CSC concluded that the IBM Watson services was unnecessary convoluted for the client's intended purpose.

In comparison, AWS's myriad of cloud solutions facilitated for development of various applications throughout the spectrum. As a result, AWS solutions can be easily designed to meet varying levels of business needs as the company grows [35, 38]. This inherently reflects AWS's flexibility, scalability (Requirement A10) and ease of deployment in cloud computing design [35]. Based on these differences, CSC asserted that AWS's foundations were more suitable for the WSN. This decision has been justified by AWS's superior aggregate rating of 86.5 as seen from the weighted decision matrix from Table 21.

Table 21: CSP Weighted Decision Matrix

Criteria	Weighting	AWS Score	AWS Weighted Score	IBM Watson Score	IBM Watson Weighted Score
Suitability	30	80	24	70	21.0
Development Costs	15	90	13.5	90	13.5
Documentation	15	90	13.5	90	13.5
Flexibility	15	80	12.0	70	10.5
Support	15	90	13.5	90	13.5
Modularity	10	100	10	70	7.00
Aggregate Rating			**86.5**		**79.0**

5.3.3 Choice of Database

CSC has recognised that the cloud's ability to store and process large datasets as a critical characteristic since the supply of data is currently growing at an unprecedented rate. Increasing in volume, variety, variability and velocity, it has brought forth a phenomenon called 'Big Data' [39, 40]. It was recognised that the IoT based WSN would likely encounter the challenges of 'Big Data'.

Through a comprehensive analysis, it was determined that both relational and unstructured databases could sufficiently meet the relevant mandatory requirements as they both had the capability to timely upload their datasets (Requirement M06, requirement M09). Therefore, CSC investigated possible solutions to maximise the performance and quality of the database.

In analysing the aspirational requirements, CSC highlighted the importance of the practical and flexible databases. In comparing relational and unstructured databases, it was clear NoSQL databases were easier to manage and quicker to deploy which is derived from its schema-less characteristics. In addition, NoSQL databases also facilitates for faster data transactions between the cloud repository, embedded manager and web application (Requirement M06).

Furthermore, the schema-less characteristics of NoSQL databases allows new datasets to be easily stored since there are no schema limitations, satisfying the aspirational requirement of configuration flexibility (Requirement A03). The holistic decision to incorporate NoSQL database into the cloud architecture has been justified by its superior aggregate rating of 80 from the weighted decision matrix in Table 22.

Table 22: Database Weighted Decision Matrix

Criteria	Weighting	NoSQL Database	NoSQL Database Weighted Score	Relational Database	Unstructured Database Weighted Score
Configuration Flexibility	60	90	54	60	36
System Costs	30	60	18	60	18
Scalability	10	80	8.0	70	7.0
Aggregate Rating			**80**		**61**

5.3.4 Database Design

Naturally the selection of AWS and the NoSQL database has resulted in the implementation of Amazon DynamoDB, AWS's multipurpose NoSQL database for IoT systems [35]. Amazon DynamoDB has been purposely designed to be "always writable" which translates to a system that minimises data losses [41]. Historically this has been substantially effective in web-applications which require data aggregates [35]. CSC has asserted that the "always writable" functionality is an extremely valuable asset to the client as it reflects on the aspirational requirement for scalability (Requirement A10) [41].

In utilising Amazon DynamoDB for the WSN, CSC has created a NoSQL database comprised of two tables storing data samples and configuration settings respectively. CSC has named these tables "Sensor Data Table" and "Configuration Data Table" respectively.

DynamoDB operates on a simplistic model which is comprised of three main components – tables, items and attributes. The DynamoDB table refers to the overarching collection of information where all the of the respective datasets are stored. In each of these tables, there are multiple unique items representing a group of attributes. These attributes are fundamental data elements and cannot be broken down further [35] [42]. DynamoDB's simplistic model for data storage has significant advantages as it functions on the premise of storing ad-hoc dictionaries. Consequently, this allows the client to freely change the contents of their data uploads (Requirement A03). This translates to a reduction in system costs as it reduces database management costs (Requirement A07).

When creating the database, it is paramount to specify a primary key. The primary key is a unique identifier which is used to index the data collection. Due to the client's intention to query data based on mote identification numbers and timestamps, CSC has diligently incorporated a composite primary key for both the "Sensor Data Table" and "Configuration Data Table" [34]. In using composite keys, stored data is indexed based on the values of the partition key and sorted by its sort key which is depicted in Figure 41.

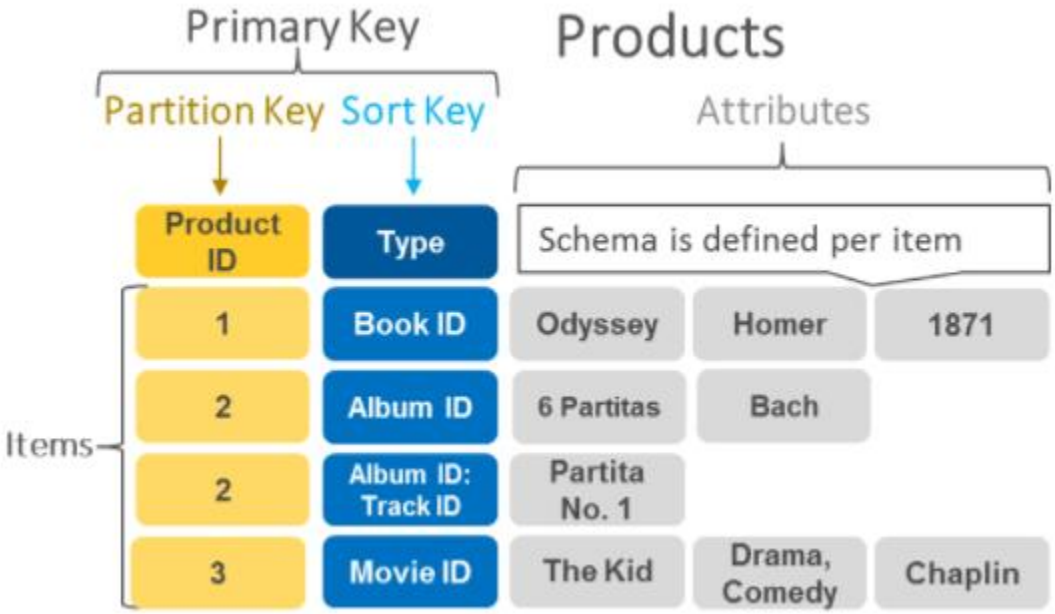


Figure 41: Composite Keys

Since the schema-less characteristics of DynamoDB presents a trade-off between sagacious flexibility in data writes and query ability, it is paramount to optimise the configurable parameters [35] [42]. Therefore, the fidelity of the database is highly correlated to the effective selection of the composite key. Intuitively since DynamoDB does not support complex queries, it is imperative to ensure that the high-volume queries are as simplistic as possible. CSC has postulated that the client will mainly use the "Sensor Data Table" to service a web application requiring queries of the most recent sensor data based on mote identification numbers. Therefore, the Sensor Data Table has been indexed by mote identification numbers (partition key) and sorted (sort key) by timestamps.

The deployment of the "Sensor Data Table" will store sensor data attributes in item schemas and is expected to easily cope with varying types of datasets such as temperature, velocity, acceleration and position (Requirement A03). The "Sensor Data Table" repository has been visualised in Figure 42 and displays DynamoDB's composition of composite keys, items and attributes.

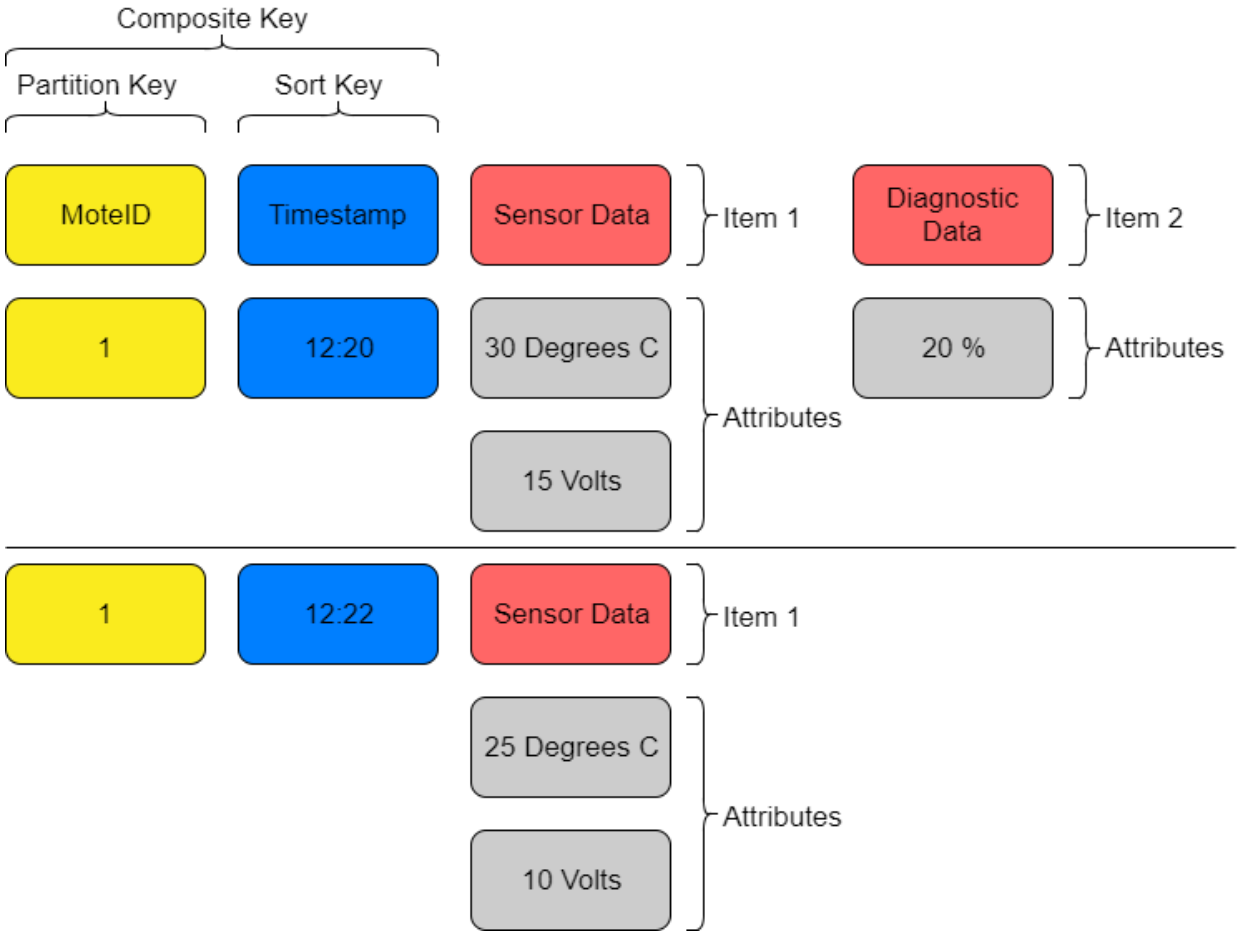


Figure 42: Sensor Data Table Repository

From CSC's analysis of Amazon DynamoDB's storing methods it was concluded that the "Sensor Data Table" will be rapidly populated therefore adversely effecting query speeds. To improve database performance, CSC has proposed a 'store and dump' procedure which redistributes low-value data into bulkier, cheaper data warehouses such Amazon Simple Storage Service [35]. This will effectively allow the client to preserve the entirety of their data whilst facilitating for reasonable query times. Typically, time-series databases often incur high data transactions for the most-recent datasets whilst rarely using datasets from the past [35]. Therefore, CSC will 'dump' the 'old' datasets from the Sensor Data Table on an ad-hoc basis, thus optimising processing time.

The deployment of the 'store and dump' procedures incur additional project constraints which arises from unintentionally partitioning important datasets. Through client consultations, CSC determined that the configuration data could be accidentally partitioned (dumped) due to the system's infrequent configuration writes [34]. CSC has resolved this issue by deploying the "Configuration Data Table". Similarly, CSC has chosen to implement a composite primary key composed of a mote identification number partition key and a timestamp sort key to optimise the query performance of the system. The "Configuration Data Table" repository has been visualised in Figure 43 and is intended to store respective information of location and sampling rates [34].

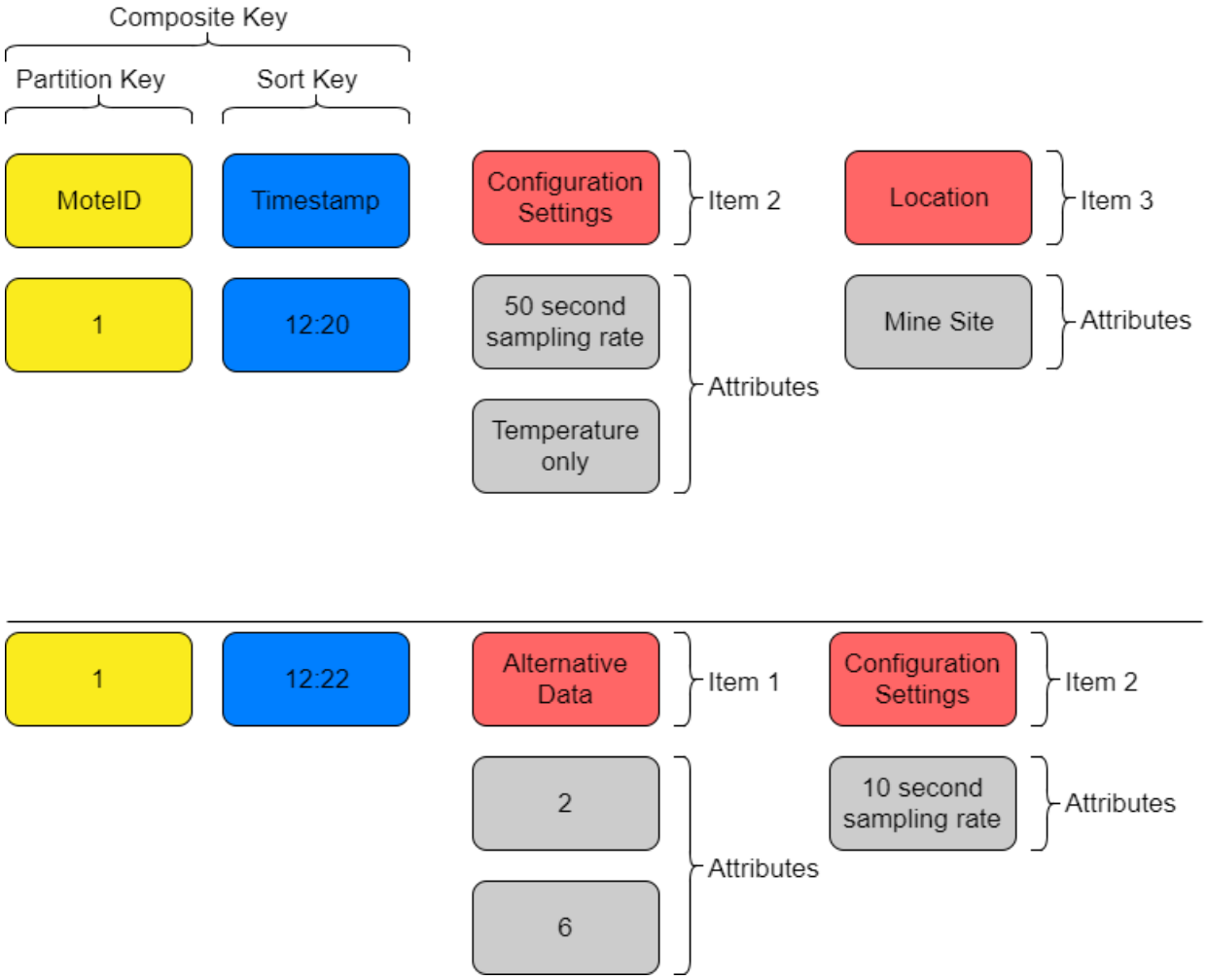


Figure 43: Configuration Data Table Repository

5.3.5 Web Application Framework

A web application framework (WAF) is a software skeleton intended to support the development and maintenance of web applications [44]. This is through automating the overhead associated with common activities performed in web development, such as providing libraries for database access, template frameworks, and session management. The chosen WAF heavily dictates the implementation and design of the web application itself. As different frameworks are often written in different programming languages, it is also difficult to transition between frameworks post-implementation [44], emphasising the importance of an initial sagacious choice.

The most popular and well supported WAFs are Django [45], Ruby on Rails [46], and Google App Engine [47]. Although comparable, there are distinct deviations which are relevant to the selection of the ideal WAF for this project. The primary requirements pertinent to the selection of a WAF are M03 (design strategy), A08 (amplified user experience), as well as integration considerations with the chosen cloud database. In the context of WAFs these can be translated into the framework's documentation, configurability, and integration. Detailed documentation enables a clearer design strategy. This is facilitated through CSC's increased comprehension of the software tools being used. In addition, it eases client maintenance to the user interface in the absence of CSC post-project. Configurability permits CSC to provide an amplified user experience through providing the means to precisely design the user interface. Integration with the cloud database is an additional consideration, as the web application must align with the rest of the design.

A weighted decision matrix has been used to aid design decisions (Table 23). Weightings sum to 1; criteria is ranked from 0 (lowest) to 100 (highest) for each of the WAF. Weighting values are derived from the project requirements; ranking is defined through consultation with the literature.

Table 23: WAF Weighted Decision Matrix

Criteria	Weighting	Django	Ruby on Rails	Google App Engine
Documentation	0.5	70	80	60
Configurability	0.2	80	70	60
Integration	0.3	90	40	50
Weighted Sum		**78**	66	57

CSC has discerned that Django is the ideal choice for the WAF. Although there is slightly less documentation than Ruby on Rails, there is still ample information available [48]. In addition, Django outperforms Ruby on Rails and Google App Engine with respect to configurability and integration. Django integrates with the chosen database (AWS) [49, 50] in addition to having a high level of configurability [45]. Hence Django will be chosen as CSC's WAF.

5.3.6 Interface Design

From Table 23 CSC discerned that Django is the ideal choice for the WAF. Although there is slightly less documentation, a crucial component of the web application is the visual design of the GUI [51]. The GUI pertains to requirements M10 (client application), A05 (authentication), and A08 (amplified user experience). The GUI must achieve a balance between simplicity and control – there must be enough control for the user to achieve what they want from the interface, without the unnecessary clutter of additional controls. Non-essential functionality superfluously wastes resources while being visually displeasing to the user. CSC has designed the user-interface under the guidance of these key principles.

Users should only be asked to authenticate in exchange for value, and sign-in should be delayed for as long as practicable to avoid unnecessarily obstructing the user [52]. As this web application acts as the interface to potentially confidential and/or commercially sensitive information, users must first authenticate themselves before being granted any further access to the application. When a user accesses the client's main web domain, they are greeted with the interface shown in Figure 44**.** The simplicity of the login page is intentional - the sign-in verification process should be quick and discreet to minimise its detraction from the application [51, 52]. Best practices for a login screen have been employed [51, 52]: the layout is centred; elements are aligned; and all unnecessary noise has been removed. The area for user input has a shadow and slight contrast in colour to the web page's background, employed to subtly draw the user's attention to the pertinent area for authentication. A username and password method for authentication was chosen to align with the client's requirements (Requirement A05). The inputs for the username and password have been pre-populated with the slightly transparent text "Username" and "Password" - which disappear when the user beings typing in these areas. This communicates to the user the correct location to input their credentials while minimising the clutter of the interface.
.

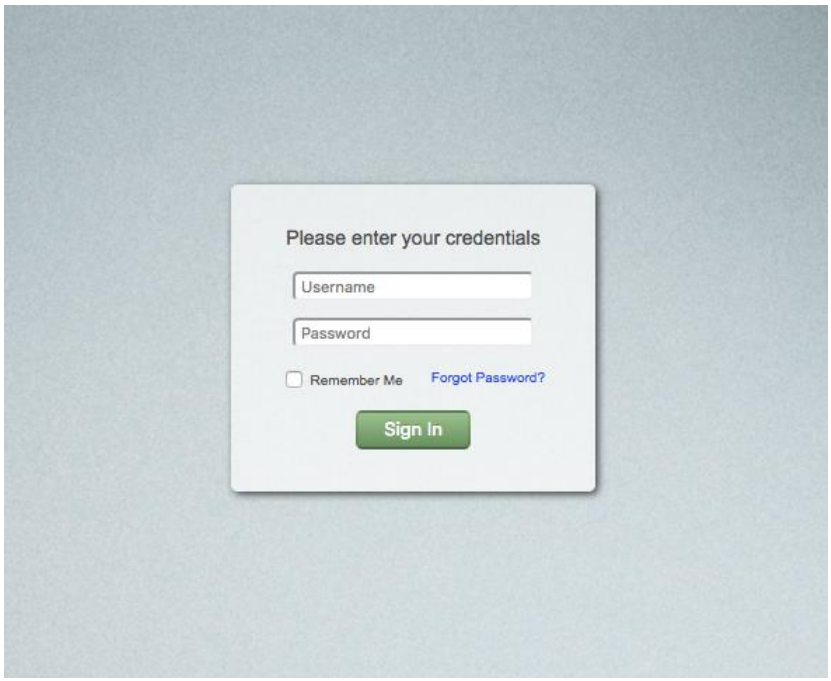


Figure 44: Login Screen

Hypertext is text on a screen that has references (or hyperlinks) to other text on different web pages that the user can immediately access via clicking on this text [51]**.** Hypertext has been marked in blue – the de-facto standard for hypertext [53] – to appeal to the user's intuition garnered through previous experience browsing the world wide web. The exception to this is the "Sign In" link, which has instead opted for an aesthetically appealing button to attract the user's attention (Requirement A08). This use of a "Sign In" button is standard amongst login screens [52]; as signing in is the primary functionality of login screen, the larger button is used to draw attention to an area of the screen frequented when accessing the page. A shadow will appear behind the button when hovered over by the user - further communicating its interactivity and adding to the aesthetics of the overall web page design (Requirement A08) [52]**.** If a user attempts to sign in with credentials that do not match those of existing username and password combination in the database, they receive an error message "*The username and/or password you specified are not correct."* **(**Figure 45**).** Although the authentication system is intelligent enough to determine which of these two fields is incorrect, it intentionally does not specify which for security reasons. Giving insight into the incorrect field significantly reduces the sample space of possible username and password combinations, increasing the probability of granting access to an unauthenticated user [54]. The error message uses red font to render it visually distinctive, and the wording was chosen to be both helpful and (appropriately) precise [51].

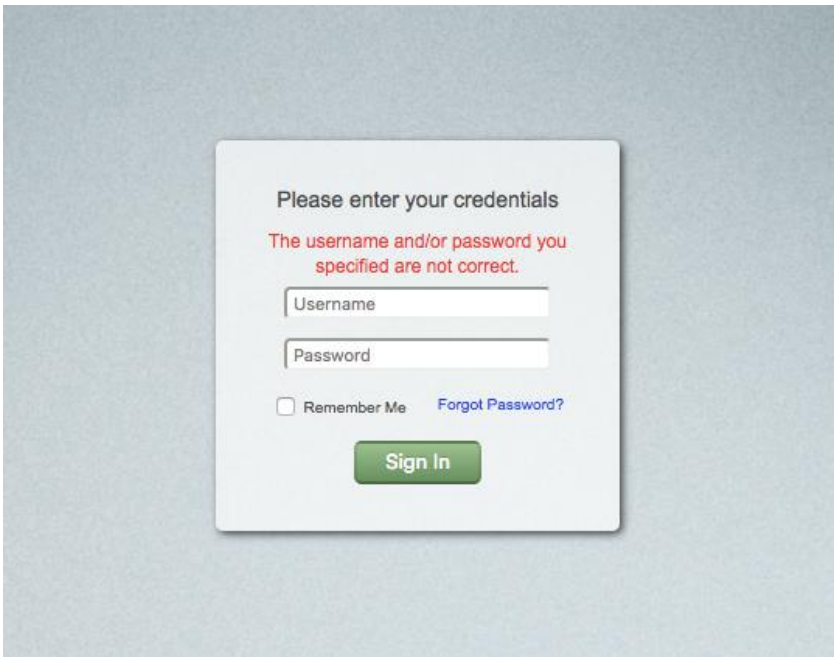


Figure 45: Login Screen Error Message

Once authenticated, users are immediately permitted to access the web application (Figure 46). The upper portion of the page has been reserved to display the logo and other identification information pertinent to the user, as desired by the client (note that the client's logo has been used for demonstration purposes). The section beneath this is comprised of four primary elements that facilitate interaction with the time-series data display: the time resolution buttons; range slider; legend; and toolbar. These elements provide the user with a rich level of interactivity while maintaining an intuitive and simplistic design.

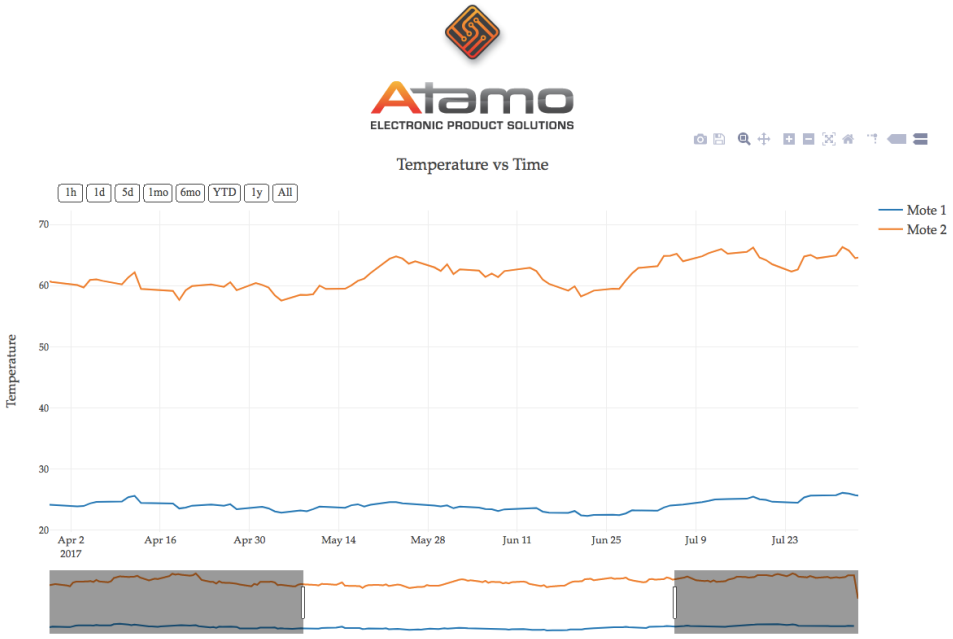


Figure 46: Web Application Post-Login Screen

The time resolution buttons allow the user to alter the time period across which data is displayed through clicking a button corresponding to a fixed time period - 1 hour, 1 day, 5 days, 1 month, 6 months, year to date, 1 year, and All.

The time periods were chosen based on the client's requirements for hourly, daily, monthly, and yearly display of data; the buttons provide convenience and simplicity for a user in selecting a desired time period. The time resolution may also be altered using the range slider residing underneath the x-axis. This allows for a more specific level of granularity in the display of data if desired. The portion of the data currently not displayed on the graph is darkened in the range slider to communicate to the user the currently highlighted section. The start and end date of the displayed data may be altered through clicking and dragging on the left and right range slider handles respectively; the slider enables the user to alter the axis to choose a custom start and end date. Clicking and dragging on the currently highlighted section of the range slider allows the user to keep the time period constant while shifting through the displayed data (Figure 47, Figure 48)

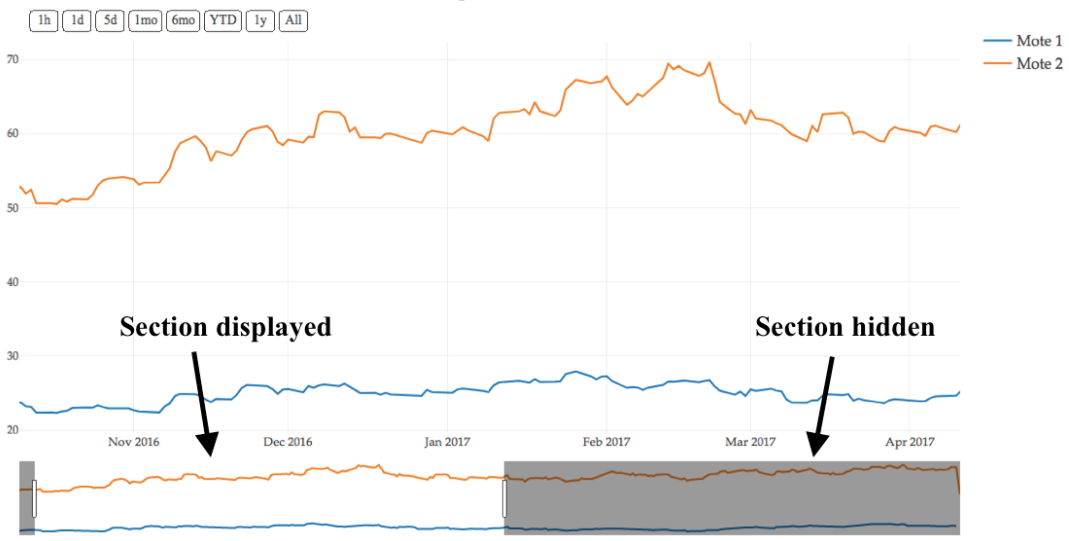


Figure 47: Range Slider Illustration: Initial Selection

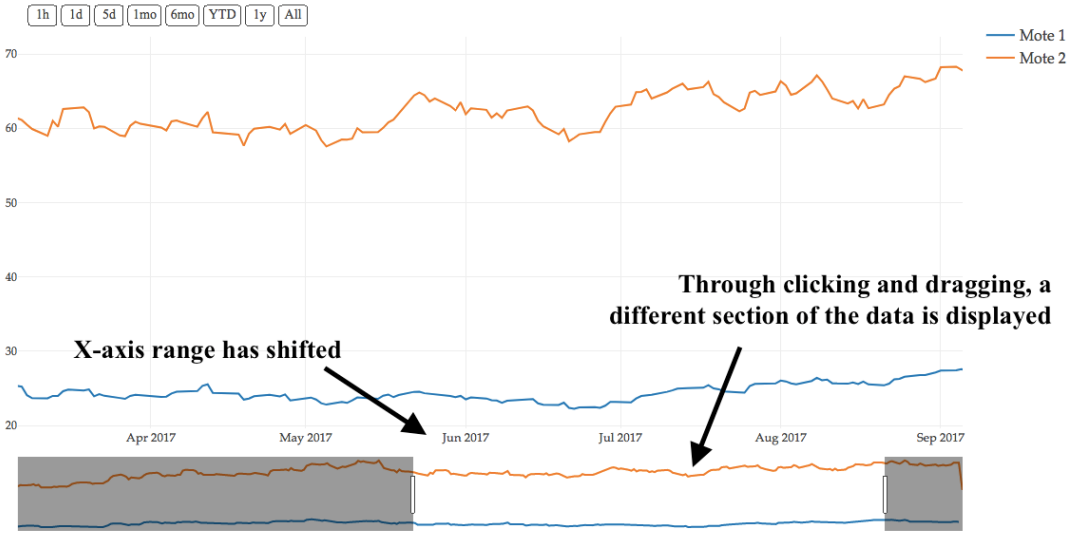


Figure 48: Range Slider Illustration: Altering displayed data

The legend on the right hand side of the graph matches each trace's colour with its appropriate label. This legend offers a level of interactivity: the user is able to click an individual item in the legend to select/deselect the appropriate trace. Only the selected traces will be displayed on the graph. By double clicking an item in the legend, all other items will be deselected with only the chosen trace displayed on the graph - providing the user the means to

easily isolate the data from a single mote. As this double clicking behaviour is not necessarily intuitive, a notification reading "*Double click on legend to isolate individual trace"* will appear when first clicking on the legend (Figure 49). To avoid unnecessarily obscuring the interface, subsequent clicks will not refresh this notification. The motes that are currently deselected will appear "greyed out" on the legend (see "Mote 1" in Figure 49) - providing the user with an intuitive signal as to which items are currently being plotted.

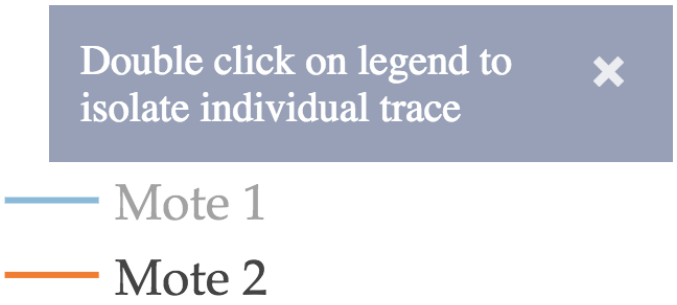


Figure 49: Legend Notification

When a legend item is selected/deselected, the range of the y-axis will dynamically adjust to more accurately fit the data that is desired for display (i.e. the remaining selected traces after a user has selected/deselected a legend item). Figure 50 depicts the graph with both "Mote 1" and "Mote 2" selected; Figure 51 depicts the graph immediately after deselecting "Mote 2". It can be observed that the y-axis has dynamically scaled its range from 20-70 in Figure 50 to 22-28 in Figure 51. This auto scaling significantly increases the readability of the interface (Requirement A05), facilitating more granular insights into the displayed data. Note that the user is not required to refresh the page to realise this functionality.

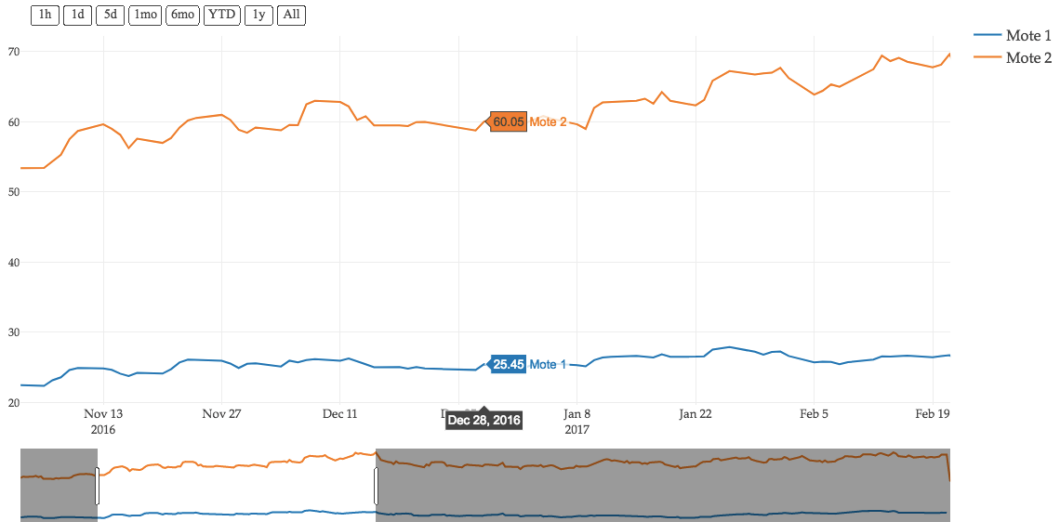


Figure 50: Legend illustration - Both Motes selected

Figure 51: Legend illustration - Single Mote selected

The toolbar (Figure 52) provides the user with extended functionality in their interactive with the interface. Hovering over any of the icons will indicate the specific tools functionality. The tools allow the user to download the plot as an image, set zoom, zoom in or out, autoscale, pan, reset axis, toggle spike lines, show closest data on hover, or compare data on hover.

Figure 52: Graph Toolbar

The 'set zoom' allows the user to highlight a particular area of interest displayed on the graph by clicking and dragging across the pertinent area (Figure 53). After releasing the pointer, the interface will then adjust the range of the x-axis to match the area highlighted (Figure 54). This interaction provides another means for the user to easily pinpoint a specific section of the graph for further analysis.

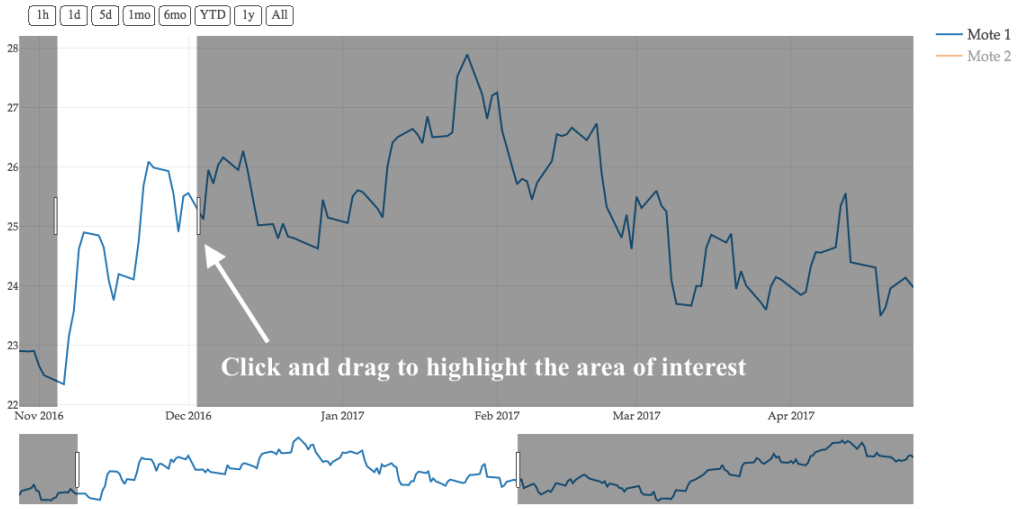


Figure 53: Set Zoom Illustration - Click and drag

Figure 54: Set Zoom Illustration - Result

The 'Toggle Spike Lines' tool provides the user with a visual guide as to a data point's x and y position. When enabled, hovering over a data point will produce dashed lines running from the data point to the x and y axis (Figure 55). This information can be made further verbose through selecting the "Show closest data on hover" tool, which will display verbose identification of a data point when hovered over - (*"Feb 2, 2017, 26.61") Mote 1* in Figure 55. Alternatively, enabling the "Compare data on hover" tool will show identification information for all selected traces when hovering over a particular time period on the graph (Figure 50).

Figure 55: Toggle Spikes Enabled

The aforementioned details describing the user interface demonstrate CSC's achievement of the requirements M10 (client application), A05 (authentication), and A08 (amplified user experience).

5.3.7 Deployment

Software deployment refers to the multifarious activities that facilitate a software system's availability for use [55]. This includes release, adaption, capacity provisioning, scaling, and system maintenance. Deployment pertains to requirements M03 (design strategy), A08 (amplified user experience), and A10 (database scalability); deployment is

heavily contingent on the chosen cloud database. Software designed to ease capacity provisioning and scaling is often specific to the cloud infrastructure – frequently offered as an on-sell from the database supplier [49, 56]. Although other software deployment is still possible, the added complexity and lack of supporting documentation does not align with requirement M03. To bolster this, AWS offers a deployment service – Elastic Beanstalk – which handles the details of capacity provisioning, load balancing, scaling, and application health monitoring [50]. With amply documentation, this service meets requirements M03, A08 and A10.

A similar software offered by AWS is known as CloudFormation [57]. Albeit well documented, it requires an advanced knowledge of AWS infrastructure to maintain, and takes a significantly longer time to deploy [50, 57]. Hence to prevent the project becoming protracted as well as ease upkeep post-project, Elastic Beanstalk is chosen to facilitate software deployment of the web application.

6 Testing

The approach taken to testing in this project is based on three types of testing

1. Unit testing,
2. Integration tests, and
3. System tests

Unit tests are defined as tests of discrete, non-subdividable design elements. Integration tests cover situations where more than one unit, and in particular interfaces between units, are being tested. System tests encompass an entire system with multiple units and interfaces.

Note that in the case of testing Embedded Design components, a testing suite has been built into the duinoPRO software. Tests can be selectively enabled via macros at the user's discretion at build time. This was chosen to ensure the compiled code is sufficiently small to fit within the duinoPRO's limited memory (constraint C01).

Due to time constraints, no system tests were completed. However, a significant amount of unit testing has been successfully completed, with some components of the design progressing as far as integration testing. All tests carried out for this project are fully recorded in the ELEC5552 Team 14 Testing Document [57]. The following sections give a brief overview of the testing that has been carried out and some further testing plans.

6.1 Unit Tests

Unit testing of several components of the overall system has been successfully completed. These tests are summarised in Table 24.

Table 24: Unit Tests Summary

System Component	Relevant test(s)	Test outcome(s)
Pin mapping for Dusty-duinoPRO	TU_SH/DpUart_External_Op	Passed
	TU_SH/DnJoin_Cli_Op	Passed
Mote join routine	TU_SH/DnJoin_Cli_Op	Passed
Sampling	TU_SH/DpFrameHdr_PayloadHdr_Op	Passed
	TU_SH/DpReserveField_Payload_Op	Passed
	TU_SH/DpSample_SampleTime_Op	Passed
	TU_SH/DpSample_SampleDiag_Op	Passed
	TU_SH/DpSample_SampleSens_Op	Passed
Network manager & gateway	TU_NM/WSNHandshake_APIConnNoSerialMux_Op	Passed
Database	TU_DB/Put_Exclusive_Op	Passed
	TU_DB/Get_Exclusive_Op	Passed
Database to Web Application interface	TU_CI/DataTransfer_S3_Op	Passed
Web Application/GUI	TU_SH/ Load_Webpage_Op	Passed
	TU_SH/User_Authenticate_Op	
Other – development tools and checks	TU_SH/DpSys_FlashBlink_Op	Passed
	TU_SH/DnSys_HwareCheck_Op	Passed

For the Embedded Design, unit testing is achieved with an abstract class, Test-Case, that exposes the virtual method "run". Developers implement their unit-test by deriving from the Test-Case class and implement the run method as required. Standardised success and failure routines are provided in the abstract class, and the option to stop on success or continue is provided. Some units (most notably, the Dusty/WSN related units) cannot be tested 'as units' and are only tested in the integration phase.

6.2 Integration Testing

Limited integration testing has been completed for certain parts of the overall system. These are listed in Table 25.

Table 25: Integration Tests Summary

System Component(s)	Relevant test(s)	Test outcome(s)
Dusty Module, duinoPRO – Mote join routine	TI_SH-NM/DnJoin_Mode4_Op	Passed
Sampling	TI_SH/DpSample_SampleMain_Op	Passed
Database to Web Application interface	TI_DB-WA/Connection_Mock_Op	Passed

In addition to these completed integration tests, the following tests listed in Table 26 are recommended for testing during further development of the system. Note that Business-As-Usual (BAU) cases are only mentioned if non-trivial outcomes are expected.

Table 26: Integration Test Plan

Unit of Concern	Case	Expected Outcomes
WSN-Join *(Requires Network Manager)*	Network present, on-start attempt (BAU)	BAU
	Network present, but connection fell mid-operation, system to attempt re-join	BAU + Check scheduler is capable of disrupting routines to prioritize re-join
	Network present, connection fell during sleep, system to attempt re-join	BAU + Check scheduler is capable of detecting disconnect post-sleep
	Network not present, (on-start attempt & post-disconnect)	Check system locks in joining state and does not release until successful join. Further check system follows energy optimization routine (increased wait-time between attempts)
WSN-Push *(Requires Network Manager)*	Connected, periodic sample (BAU)	BAU
	Connected, NACK	Check that scheduler prevents lock with timeout. Check re-attempt behaviour.
	Disconnected	Check system locks into joining state and does not push.
	Post-Disconnected recovery	Check system correctly samples new data (and does not re-attempt to sample old, passed data)

Unit of Concern	Case	Expected Outcomes
Sensor-Read *(Requires sensor)*	BAU	Check accuracy
	Input validation	Check input buffers do not cause system fault. Check recovery/handling for future samples.
	Incorrect driver	Check system enters critical error state, and network is notified
Config-Set *(Requires Network Manager)*	Bad payload; incorrect field description causing the remaining portion of payload to be un-parsable	Check configuration is not updated if a bad payload is given
	Input validation	Check configuration prevents bad inputs for all parameters (as specified in LUT). Check configuration is not set out-of-range
SYS-Sleep	BAU	Check power consumption, accuracy of sleep time, shutdown time.
	Sleep-guard lock timeout (waiting on ACK, configuration handling)	Check sleep guard does not lock system for certain scenarios after configured anti-lock timeout expires.
	Sleep-guard lock (BAU)	Check sleep guard set by external unit prevents system from sleeping
SYS-Wake	BAU	Check recovery time; check all units are responsive
	Failure on one or more units (simulated with intervention)	Check system recognition of failure and re-attempt

6.3 System testing

As mentioned, no system testing has been completed due to time constraints. However, in completing potential future system tests, an end-to-end approach should be used, as the integration tests will cover all cases of failure/operation. Key to the end-to-end test is ensuring the two primary transactions from Section 0 can be achieved.

6.4 Testing Summary

In summary, Table 27 lists the various components of the system and lists their status in regard to design, implementation and testing, as well as the relevant requirements addressed in testing these.

Table 27: Development & Testing Summary

System Component	Designed	Implemented	Tested	Requirements
Pin mapping for Dusty-duinoPRO	✓	Partial	Partial	M05, M12, A02
Mote state management	✓	×	×	M04, A04, A06
Mote sleep management	✓	×	×	M04, A06
Mote main routine	✓	×	×	M04, M07, A04, A06
Mote join routine	✓	Partial	Partial	M05, A02
Sampling	✓	✓	✓	M07, M08
Data acquisition	✓	✓	×	M07, M11, A01
Sensor configuration	✓	×		A04
Mote configuration	✓	×	×	A03, A04
Mote to Network Manager interface	Partial	×	×	M05, M09, A02
Network manager & gateway	✓	Partial	Partial	M05, M06, M08, A02
Network Manager to Database interface	✓	×	×	M06, M09
Database	✓	✓	✓	M06, A10
Database to Web Application interface	✓	✓	✓	M09, M10
Web Application/GUI	✓	✓	✓	M10, A05, A08

7 Resources

Due to the complex nature of the architecture, CSC has leveraged a multitude of resources to assist in the design and implementation of the project. To maintain transparency and quality assurance, the team leveraged GitHub to establish a version control system for all documents used throughout the project [58].

7.1 Hardware

PCB design has been achieved using EagleCAD, enabling the team to produce a schematic and layout that aligns with electrical protocols and manufacturing standards [59]. EagleCAD was chosen due to the team's previous experience with the software and the portability of the final design. EasyEDA, a web-based electronic design automation tool, was selected for the fabrication of the PCB due to its ease of use and competitive pricing [60].

7.2 Network

The team has selected a DC2274A-A device to act as the Network Manager. The device was chosen for to its compatibility with the Smartmesh IP that our wireless sensor network implements, as well as the portability associated with a bidirectional USB connection to the gateway [61]. A Raspberry Pi will act as the gateway between the network manager and the database due to its compact size and cost.

7.3 Front End

CSC has selected AWS as the cloud services platform for the front-office portion of the project. AWS promotes compatibility with many software applications that are able to leverage for cloud-based tasks [62]. Implementation of the database will be achieved using DynamoDB, Amazon's NoSQL database service that handles the routing of data requests, enabling the user to scale the dataset while still maintaining speed and reliability [63]. Development of the GUI will be achieved by utilising AWS Elastic Beanstalk, enabling simple management of individual applications in the AWS Cloud without having to interface the infrastructure behind each application [64].

8 Risks & Contingencies

Risk and contingency management includes identifying, controlling, and mitigating potential risks and contingencies [65]. Risk is an inevitable part of all stages of the project lifecycle, although proper management in the planning phase can minimise risk frequency and impact. In line with this, CSC performed a risk assessment in the form of a risk register, identifying the top five risks to the project (Table 30 & Table 31).

Multiplying the risk likelihood and consequence gives a single metric by which to evaluate the significance of each risk, shown in Table 28.

Table 28: Risk Ranking Matrix

		Consequence				
		Minor	*Medium*	*Serious*	*Major*	*Catastrophic*
Likelihood	*Almost Certain*	Moderate	High	Critical	Critical	Critical
	Likely	Moderate	High	High	Critical	Critical
	Possible	Low	Moderate	High	Critical	Critical
	Unlikely	Low	Low	Moderate	High	Critical
	Rare	Low	Low	Moderate	High	High

Four levels of risk have been used in the risk assessment, outlined in Table 29.

Table 29: Risk Rank Defined

Risk Rank	*Explanation*
Critical	Risk is not acceptable. Immediate action must be taken to eliminate the risk, or implement measures to lower the risk to an acceptable level.
High	Risk is tolerable; action is required. Identify and implement controls to reduce the risk in accordance with the principles of As Low As Reasonably Practicable (ALARP).
Moderate	Risk is tolerable. Action is required. Identify and implement controls to reduce the risk in accordance with the principles of ALARP.
Low	Risk is acceptable.

Table 30: Risk Register Part 1

No.	*Hazard Description*	*Impact Description*	*Inherent Risk*			*Mitigation / Control Measures*	*Residual Risk*			*Responsibility*
			Consequence	*Likelihood*	*Risk*		*Consequence*	*Likelihood*	*Risk*	
1	Incorrect termination of a pin required for a critical dusty module feature.	Significant time delays to project; possibility of unmet deliverables.	Major	Unlikely	High	Develop a working prototype before termination of pins; robustly developed embedded software application; require client sign off on PCB design.	Serious	Rare	Moderate	CSC
2	Testing environment does not mimic real world operating conditions.	Systematic underrepre-sentation of expected battery lifetime; reduced field operating time.	Serious	Possible	High	Robust survey of the literature to procure practical weather estimates; over designing to withstand conditions above those expected.	Medium	Unlikely	Low	CSC in design, the client in defining the operating conditions
3	Inappropriate handling of electronics damages components.	Minor injuries to team member(s), significant time delays to project; possibility of unmet deliverables.	Serious	Unlikely	Moderate	Induction with qualified lab demonstrator; proper use of equipment; antistatic wrist strap and/or safety glasses where appropriate.	Medium	Rare	Low	CSC

continued ...

Table 31: Risk Register Part 2

...continued

No.	*Hazard Description*	*Impact Description*	*Inherent Risk*			*Mitigation / Control Measures*	*Residual Risk*			*Responsibility*
			Consequence	*Likelihood*	*Risk*		*Consequence*	*Likelihood*	*Risk*	
4	Tight time restrictions on required deliverables.	Unmet project deliverables.	Major	Possible	Critical	Employ proper time management tools (e.g. Gantt chart); forming sub-teams to specialise on smaller parts of the project; regular team meetings where progress and difficulties are reported.	Medium	Possible	Moderate	CSC
5	Unmonitored Amazon resources.	Significant accumulation of billable cloud resources.	Medium	Possible	Moderate	Setting up Cloud Budget Alerts to email and/or text client when specified monetary cost is reached; limiting autoscaling until required.	Minor	Unlikely	Low	CSC in design, the client in implementation

9 Design Outputs

CSC has contributed to the WSN's proof of concept in a multi-faceted number of ways. This has encompassed prototyping a functional network and transparent documentation of systematic approaches in extending battery life.

Design Outputs	Purpose	Stored
Final Design Report	Clearly outline the design methodology implemented	LMS File Exchange
User Manual	Clearly outline how the client can use CSC's product	LMS File Exchange & Github
Testing Documents	Reflect the limitations of the design	LMS File Exchange
Cloud Database	Remote storage unit for the WSN	Amazon Web Services
Web Application	Medium for the client to view data directions	Amazon Web Services
Codes	Functional codes to reduce the power consumption in the WSN	GitHub
Detailed Pinout Description & Elimination	Ensure that a pin is not improperly terminated	Final Report Appendix

10 System Cost

The WSN comprises of hardware, software and human resource costs [11]. The hardware cost solely consists of the interposer PCB fabrication used to attach the Dusty to the duinoPRO. The software costs are the Asian-pacific marginal provisioned throughput rates used in data transactions. The pricing of AWS resources is based on the frequency and volume of data transactions. Therefore, the cloud-infrastructure costs have a large variability depending on the client's project size. In utilising Amazon DynamoDB, the marginal costs for the Asia Pacific region in Australia Dollars are tabulated in Table 32 along with the hardware and engineering design costs. As for this project, the cost of engineering design is found to be $89,820 where this cost varies according to the outputs and relevant activities carried out by CSC. The timesheet of CSC represents activities carried out by CSC for this project and it can be found in Appendix C.

Table 32: System Cost

	Vendor or Service Provider	Type of service	Details	Cost
Hardware	EasyEDA	PCB Fabrication	Fabricating PCB with dimension of 29.27 x 28.54mm, castellated mounting holes across the edge of the board, 2 layers board with thickness of 1.6mm and 1oz of copper.	$33.81
Software	AWS DynamoDB	Provisioned throughput cloud database storage	Write capacity unit	$0.00074/hr
			Read capacity unit	$0.000148/hr
Human Resources	CSC	Engineering design	The engineering costs pertaining to the development of the WSN	$120/hr

11 Conclusion

One of the greatest challenges in designing a wireless sensor network is the management of power consumption of the devices in the network. Nodes in a wireless sensor network are often battery powered and placed in remote locations and for these reasons energy usage must be kept to a minimum. Cloud Seven Consultants were contracted by ATAMO to investigate the power management of sensor nodes within a wireless sensor network. ATAMO also specified the use of their duinoPRO development platform and the use of Linear Technologies SmartMeshIP communication protocol. CSC demarcated this task into three functional blocks. The first of these was the hardware which involved the design of a printed circuit board to interface the dusty networking module to the duinoPRO board. The hardware design also involved the setup of the serial communication between the duinoPRO and the dusty module. The second component of the design was the embedded application responsible for completing all the tasks necessary for the duinoPRO to act as the sensor host. This included tasks such as sampling from sensors, joining the network, sending data, and the management of states (including sleep). The last part of the design was the design of a cloud database and web application to visualise the incoming data from the wireless sensor network. A gateway was also designed to act as the interface between the local WSN and the cloud database. Cloud Seven Consultants successfully produced designs for these components and in many cases, have completed successful testing to illustrate a proof of concept. Due to the tight time constraints of the project some aspects of the system design remain untested however CSC has produced plans for the ATAMO should they wish to implement the design in the future.

12 References

[1] G. Levine. (2017). *Arduino* [Image]. Available: https://visualhunt.com/photo/126201/.].
[2] National Instruments. (2016). *What is a Wireless Sensor Network?* Available: http://www.ni.com/white-paper/7142/en/. [Accessed: 25/10/2017].
[3] R. A. Abd-Alhameed, D. Zhou, C. H. See, Y. F. Hu, and K. V. Horoshenkov. (2008). *Measure The Range Of Sensor Networks*. Available: http://www.mwrf.com/test-and-measurement/measure-range-sensor-networks. [Accessed: 25/10/2017].
[4] A. Sinha and A. Chandrakasan, "Dynamic Power Management in Wireless Sensor Networks," *IEEE*, vol. 18, no. 2, pp. 62-74, 2001. Available: http://ieeexplore.ieee.org/abstract/document/914626/
[5] Linear Technology. (2017). *Smart Mesh IP*. Available: http://www.linear.com/products/SmartMesh_IP. [Accessed: Aug. 2, 2017].
[6] M. Yen, "Wireless sensor network monitors structural data and environmental contaminants," (in English), *Materials Performance*, Article vol. 47, no. 3, pp. 19-20, Mar 2008. Available: <Go to ISI>://WOS:000253665300005
[7] Y. Pang, G. Lodewijks, and Bindt, "Machinery maintenance prediction using pattern recognition of condition monitoring data," (in English), *8th International Conference on Condition Monitoring and Machinery Failure Prevention Technologies 2011, Vols 1 and 2*, Proceedings Paper pp. 772-781, 2011. Available: <Go to ISI>://WOS:000399622600078
[8] K. Ellis, S. R. Mounce, B. Ryan, M. R. Templeton, and C. A. Biggs, "Use of on-line water quality monitoring data to predict bacteriological failures," (in English), *12th International Conference on Computing and Control for the Water Industry, Ccwi2013*, Proceedings Paper vol. 70, pp. 612-621, 2014. doi:10.1016/j.proeng.2014.02.067. Available: <Go to ISI>://WOS:000341500600067
[9] A. Ta, "ELEC5552 Team 14 Minutes 170810 Week 2," 2017, Available: ELEC5552 LMS Team 14 File Exchange.
[10] M. Callaghan, "ELEC5552 Technical Queries 1-6 Week 3 20170817," 2017, Available: ELEC5552 LMS Team 14 File Exchange.
[11] P. Bouvy, "ELEC5552 Team 14 Minutes 170824 Week 4.2," 2017, Available: ELEC5552 LMS Team 14 File Exchange.
[12] ATAMO, "Close the gap with duinoPRO," n.d., Available: http://www.atamo.com.au/images/duinoPRO-ClosingtheGap-170109.pdf. [Accessed: Aug. 10, 2017].
[13] M. Callaghan, "ELEC5552 Technical Queries 7-8 Week 6 20170906," 2017, Available: ELEC5552 LMS Team 14 File Exchange.
[14] K. Clifton, "duinoPRO UNO Baseboard 3x3," 2016.
[15] M. Callaghan, "Email from Mark Callaghan on 7 August 2017," 2017, Available: ELEC5552 LMS Team 14 File Exchange.
[16] Linear Technology, "Dust Networks: Eterna Integration Guide," 2015, Available: https://cds.linear.com/docs/en/user-guide/Eterna_Integration_Guide.pdf. [Accessed: Oct. 25, 2017].
[17] Linear Technology, "Eterna Serial Programmer Guide," 2016, Available: https://cds.linear.com/docs/en/software-and-simulation/Eterna_Serial_Programmer_Guide.pdf. [Accessed: Oct. 25, 2017].
[18] Linear Technology. (2017). *LTC5800-IPM - SmartMesh IP Wireless 802.15.4e System-on-Chip* [Online]. Available: http://www.linear.com/product/LTC5800-IPM.].
[19] Atmel, "ATmega328/P Datasheet," 2016, Available: http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf. [Accessed: Aug. 22, 2017].
[20] Linear Technology, "SmartMesh IP User's Guide," 2016, Available: http://cds.linear.com/docs/en/user-guide/SmartMesh_IP_User_s_Guide.pdf.
[21] Linear Technology, "SmartMesh IP Mote Serial API Guide," Online 2016, Available: http://cds.linear.com/docs/en/design-note/SmartMesh_IP_Mote_Serial_API_Guide.pdf.
[22] ATAMO, "Getting Started Guide - duinoPRO UNO," 2016.
[23] J. Simon. (2016). *Port to Your Hardware*. Available:

https://dustcloud.atlassian.net/wiki/spaces/QSL/pages/80609405/Port+to+Your+Hardware. [Accessed: Oct. 16, 2017].

[24] STMicroelectronics, "LSM303D - Ultra-compact high-performance eCompass module: 3D accelerometer and 3D magnetometer," 2013, Available: http://www.st.com/en/mems-and-sensors/lsm303dlhc.html. [Accessed: Sept. 3, 2017].

[25] K. Clifton. (2017). *duinoPRO firmware libraries* [Online]. Available: https://github.com/duinoPRO/firmware. [Accessed: Sept. 27, 2017].

[26] ATAMO, "duinoPRO Accelerometer / Magnetometer Module," 2015.

[27] J. Phan. (2017). *Sensor Host Configurations*. Available: https://kjph.github.io/c7c-atamo-dusty/technical/arch/sh_configurations.html. [Accessed: Sept. 13, 2017].

[28] Linear Technologies. (2017). *DC2274A-A - SmartMesh IP USB Network Manager, 100 mote capacity*. Available: http://www.linear.com/solutions/5744.].

[29] Linear Technology, SmartMesh IP Embedded Manager API Guide, 2016. [Online]. Available: http://cds.linear.com/docs/en/design-note/SmartMesh_IP_Embedded_Manager_API_Guide.pdf.

[30] T. Watteyne. (2014). *SmartMesh SDK*. Available: https://dustcloud.atlassian.net/wiki/spaces/SMSDK/overview.].

[31] R. Lea, "Node-RED: Lecture 3- Basic Nodes and Flows ", 2016. [Online]. Available: http://noderedguide.com/tag/mqtt/.

[32] T. Watteyne. (2017). *SmartMesh IP and Node-RED, revisited*. Available: https://dustcloud.atlassian.net/wiki/spaces/ALLDOC/pages/110311810/SmartMesh+IP+and+Node-RED+revisited.].

[33] S. Male, "Project 14: Energy Management in Low Power Wireless Sensor Networks," 2017, Available: ELEC5552 LMS Team 14 File Exchange.

[34] J. Sacino, "ELEC5552 Team 14 Minutes 170928 Week 9.2 V3 [Design Review]," 2017, Available: ELEC5552 LMS Team 14 File Exchange.

[35] Amazon, "Amazon DynamoDB Developer Guide," 2017. [Accessed: 2017, Aug.20].

[36] International Business Machines, "How Watson Works," 2014. [Accessed: Sept. 10, 2017].

[37] International Business Machines. (2017). *Internet of Things Platform Starter*. Available: https://console.bluemix.net/catalog/starters/internet-of-things-platform-starter?env_id=ibm:yp:us-south. [Accessed: Sep.1, 2017].

[38] Amazon. (2017). *Amazon DynamoDB Pricing*. Available: https://aws.amazon.com/dynamodb/pricing/. [Accessed: Sep.1, 2017].

[39] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, "An IoT-Oriented Data Storage Framework in Cloud Computing Platform," *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, vol. 10, no. 2, pp. 1443-1452, 2014.

[40] V. Varga, K. T. Janosi-Rancz, and B. Kalman, "Conceptual Design of Document NoSQL Database with Formal Concept Analysis," *Acta Polytechnica Hungarica*, vol. 13, no. 2, pp. 229-248, 2016.

[41] D. G. Chandra, "BASE analysis of NoSQL database," *Future Generation Computer Systems*, no. 52, p. 8, 2015.

[42] G. Balasubramanian. (2017). *Choosing the Right DynamoDB Partition Key*. Available: https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/. [Accessed: Oct. 15, 2017].

[43] A. Tsalgatidou and T. Pilioura, "An overview of standards and related technology in Web Services," *Distributed and Parallel Databases*, vol. 12, pp. 135-162, 2002. doi:10.1023/A:1016599017660.

[44] A. Holovaty and J. Kaplan-Moss, "The Definitive Guide to Django," *Estados Unidos: Editorial Apress*, vol. 26, 2009. doi:10.1093/intimm/dxu027.

[45] M. Bächle and P. Kirchberg, "Ruby on rails," *IEEE Software*, vol. 24, pp. 105-108, 2007. doi:10.1109/MS.2007.176.

[46] K. Schutt and O. Balci, "Cloud software development platforms: A comparative overview," in *IEEE/ACIS 14th International Conference on Software Engineering Research, Management and Applications, SERA*, 2016, pp. 3-13. doi:10.1109/SERA.2016.7516122.

[47] Djangoproject. (2017). *Meet Django* [Online]. Available: https://www.djangoproject.com/. [Accessed: Sept. 14, 2017].

[48] Amazon Web Services. (2014). *Getting Started with AWS: Deploying a Web Application*. Available: http://docs.aws.amazon.com/gettingstarted/latest/deploy/awsgsg-deploy.pdf.].

[49] Amazon Web Services. (2016). *AWS Elastic Beanstalk: Developer Guide*. Available: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/awseb-dg.pdf.].

[50] W. O. Galitz, *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons, Inc., 2007.

[51] M. Gunderloy, *Developer to Designer: GUI Design for the Busy Developer*. Wiley, 2006.

[52] L. Green, *Technoculture: From Alphabet to Cybersex*. ECU Publications, 2002.

[53] J. Jo, Y. Kim, and S. Lee, "Mindmetrics: Identifying users without their login IDs," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 2121-2126: IEEE. doi:10.1109/SMC.2014.6974235.

[54] J. A. Forbes and E. R. Baker, "Improving Hardware, Software, and Training Deployment Processes," in *IEEE International Conference on Software Maintenance*, 2003, pp. 377-380: IEEE Comput. Soc. doi:10.1109/ICSM.2003.1235446.

[55] X. Li and Z. Cai, "Elastic Resource Provisioning for Cloud Workflow Applications," *IEEE Transactions on Automation Science and Engineering*, vol. 14, pp. 1-16, 2015. doi:10.1109/TASE.2015.2500574.

[56] Amazon Web Services. (2010). *AWS CloudFormation User Guide*. Available: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-ug.pdf.].

[57] Cloud Seven Consulting, "ELEC5552 Team 14 Testing Document," 2017, Available: LMS File Exchange.

[58] GitHub. (2017). *How Developers Work*. Available: https://nodered.org/about/. [Accessed: Sept. 14, 2017].

[59] Autodesk. (2017). *PCB Layout Software for Every Engineer*. Available: https://www.autodesk.com/products/eagle/features. [Accessed: Sept. 14, 2017].

[60] Easy EDA. (2017). *An Easier EDA Experience*. Available: https://easyeda.com. [Accessed: Sept. 14, 2017].

[61] Linear Technology. (2017). *SmartMesh IP USB Network Manager*. Available: http://www.linear.com/solutions/5744. [Accessed: Sept. 14, 2017].

[62] Amazon. (2017). *Cloud Computing with Amazon Web Services*. Available: http://www.linear.com/solutions/5744. [Accessed: Sept. 14, 2017].

[63] Amazon. (2017). *Amazon DynamoDB Documentation*. Available: https://aws.amazon.com/documentation/dynamodb/. [Accessed: Sept. 14, 2017].

[64] Amazon. (2017). *What Is AWS Elastic Beanstalk?* Available: https://nodered.org/about/. [Accessed: Sept. 16, 2017].

[65] S. Hartley, *Professional Project Management: Bridging Complexity, Uncertainty and Change*, 2 ed. TUP Textbooks, 2016.

13 Appendices

13.1 Appendix A – Yet to be added

13.2 Appendix B – IoTeam Dusty Net List

	Pin #	I/O	Pull	Active	Description	Net	Justification
AI_0	15	I	-	-	**Analog Inputs.** These pins are multiplexed to the analog input chain. The analog input chain, as shown in Figure 12, is software-configurable and includes a variable-gain amplifier, an offset-DAC for adjusting input range, and a 10b ADC. Valid input range is between 0 to 1.8V. Analog inputs can be sampled as described in the On-Chip Software Development Kit (OnChip SDK).	3V3 via 10kΩ	Input voltage is stabilized and influence of noise caused by power supply is decreased. Resistor is required to place as close as possible to the pin. If the distance between resistor and pin are too far apart, the long wiring acts adversely as an antenna and may result unwanted EMI.
AI_1	16	I	-	-			
AI_3	17	I	-	-			
AI_2	18	I	-	-	.		
RESETn	22	I	Up	LOW	**RESETN** RESETn and FLASH_P_ENn asserted during in-circuit programming. The asynchronous reset signal is internally pulled up. Resetting Eterna will result in the ARM Cortex M3 rebooting and loss of network connectivity. Use of this signal for resetting Eterna is not recommended except during power-on and in-circuit programming.	Switch between 3V3 via 10kΩ and GND	Held high to prevent assertion (low)
TDI	23	I	UP	-	**JTAG Test Data In** **JTAG Test Data Out** **JTAG Test Mode Select** **JTAG Test Clock** JTAG Port Supporting Software Debug and Boundary Scan. An IEEE Std 1149.1b-1994 compliant Boundary Scan Definition Language (BDSL) file for the WR QFN72 package can be found	N/C	Internally pulled up
TDO	24	O	-	-		N/C	Outputs do not required tie
TMS	25	I	UP	-		N/C	Internally pulled up

	Pin #	I/O	Pull	Active	Description	Net	Justification
TCK	26	I	DOWN	-		3V3 or GND	Internally pulled down
DP4	27	I/O	-	-	**General Purpose Digital I/O (GPIO23)**	3V3 via 10kΩ	Input Mode: See AI_0 Output Mode: Disable in Flash and left unconnected
DP3 TIMER8_EXT	33	I/O I	- -	- -	**General Purpose Digital I/O (GPIO22)** **TIMER8_EXT** (if programmed) External input to 8-bit timer/counter	3V3 via 10kΩ	Input Mode: See AI_0 Output Mode: Disable in Flash and left unconnected
DP2 LPTIMER_EXT	34	I/O I	- -	- -	**General Purpose Digital I/O (GPIO21)** **LPTIMER_EXT** (if programmed) External Input to Low Power Timer/Counter	3V3 via 10kΩ	Input Mode: See AI_0 Output Mode: Disable in Flash and left unconnected
SLEEPN GPIO14	35	I I/O	- -	LOW -	**Deep Sleep.** The SLEEPn function is not currently supported in software. The SLEEPn input must either be tied, pulled or actively driven high to avoid excess leakage. **General Purpose Digital I/O** (if programmed)	3V3 via 10kΩ	To prevent leakage as said in description
DP0 SPIM_SS_2n	36	I/O O	- -	- LOW	**General Purpose Digital I/O (GPIO0)** **SPI Master Slave Select 2** (if programmed)	3V3 via 10kΩ	Input Mode: See AI_0 Output Mode: Disable in Flash and left unconnected
UARTC0_TX	37	O	-	-	**CLI UART 0 Transmit** **CLI UART 0 Receive** The CLI UART provides a mechanism for monitoring, configuration and control of Eterna during operation. For a complete description of the supported commands see the SmartMesh IP Mote CLI Guide.	Test Point 1	For development and troubleshooting.
UARTC0_RX	38	I	UP	-		Test Point 2	For development and troubleshooting.
SPIM_MISO GPIO11	39	I I/O	- -	- -	SPI Master (MISO) Master In Slave Out Port General Purpose Digital I/O	N/C	Disabled in Flash
IPCS_MISO	40	O	-	-	SPI Flash Emulation (MISO) Master In Slave Out	N/C	Disabled in Flash

	Pin #	I/O	Pull	Active	Description	Net	Justification
TIMER16_OUT GPIO6		O I/O	- -	- -	Port 16-Bit Timer/Counter Match Output/PWM Output General Purpose Digital I/O		
SPIM_MOSI GPIO10	41	O I/O	- -	- -	**SPI Master (MOSI) Master Out Slave In Port** General Purpose Digital I/O	N/C	Disabled in Flash
IPCS_MOSI TIMER16_EXT GPIO6	42	I I I/O	- - -	- - -	**SPI Flash Emulation (MOSI) Master Out Slave In Port** External Input to 16-bit Timer/Counter General Purpose Digital I/O	N/C	Disabled in Flash
SPIM_SCK GPIO9	43	O I/O	- -	- -	**SPI Master (SCK) Serial Clock Port** General Purpose Digital I/O	N/C	Disabled in Flash
IPCS_SCK TIMER8_EXT GPIO4	44	I I I/O	- - -	- - -	**SPI Flash Emulation (SCK) Serial Clock Port** External Input to 8-Bit Timer/Counter General Purpose Digital I/O	N/C	Disabled in Flash
IPCS_SSN LPTIMER_EXT GPIO3	45	I I I/O	- - -	LOW - -	**SPI Flash Emulation Slave Select, Active Low** External Input to Low Power Timer/Counter General Purpose Digital I/O	3V3	To prevent SPI Flash emulation; precautionary measure, this should already be disabled in Flash. Otherwise pull or tie to high.
SPIM_SS_1N GPIO 13	46	O I/O	-	LOW	**SPI Master Slave Select 1, Active Low** General Purpose Digital I/O	3V3	This should already be disabled in Flash. Otherwise pull or tie high.
SPIM_SS_0N GPIO 12	47	O I/O	-	LOW	**SPI Master Slave Select 0, Active Low** General Purpose Digital I/O	3V3	This should already be disabled in Flash. Otherwise pull or tie high.
DP1 TIMER16_EXT	48	I/O	-	-	**General Purpose Digital I/O** External Input to 16-Bit Timer/Counter	3V3 via 10kΩ	Input Mode: See AI_0 Output Mode: Disable in Flash and left unconnected

	Pin #	I/O	Pull	Active	Description	Net	Justification
PWM0 TIMER16_OUT GPIO16	49	O O I/O	- - -	- - -	***Pulse Width Modulator 0*** *16-Bit Timer/Counter Match Output/PWM Output* General Purpose Digital I/O	N/C	Pin is currently not supported in software.
SPIS_MISO UARTC1_TX 1_WIRE	50	O O I/O	- - -	- - -	***SPI Slave (MISO) Master In Slave Out Port*** *CLI UART 1 Transmit* 1 Wire Master	N/C	See **PWM0**
SPIS_MOSI UARTC1_RX GPIO26	51	*I* *I* I/O	- - -	- - -	***SPI Slave (MOSI) Master Out Slave In Port*** *CLI UART 1 Receive* General Purpose Digital I/O	N/C	See **PWM0**
SPIS_SCK SCL	52	I I/O	- -	- -	***SPI Slave (SCK) Serial Clock Port*** I2C Serial Clock	N/C	See **PWM0**
SPIS_SSN SDA	53	I I/O	- -	LOW -	***SPI Slave Select, Active Low*** I2C Serial Data	3V3	See **PWM0**
FLASH_P_ENN	55	I	UP	LOW	**Flash Program Enable, Active Low**	3V3	To prevent Flash programming
UART_RX_RTSN	66	I	-	LOW	**UART Receive (RTS) Request to Send, Active Low.** This input is always enabled and must be driven or pulled to a valid state to avoid leakage. The API UART interface includes bi-directional wake up and flow control. Unused input signals must be driven or pulled to their inactive state. In Mode 2: Transfers are initiated by Eterna asserting UART_TX_RTSn. The companion processor responds by asserting UART_TX_CTSn when ready to receive data. After detecting the falling edge of UART_TX_CTSn Eterna sends the entire packet. Following the transmission of the final byte in the packet Eterna negates UART_TX_RTSn and waits until the negation of UART_TX_CTSn before asserting UART_TX_RTSn again. The companion processor may negate UART_TX_CTSn any time after the first byte is transferred provided the time	DP::M5P1	DP Mode 2

	Pin #	I/O	Pull	Active	Description	Net	Justification
					out from UART_TX_RTSn to UART_TX_CTSn, tEND_TX_RTS to TX_CTS, is met. In Mode 4: unless the companion processor is always ready to receive a packet, the companion processor must negate UART_TX_CTSn prior to the end of the current packet. Failure to negate UART_TX_CTSn prior to the end of a packet may result in back to back packets. Third, the companion processor must wait at least $t_{RX-INTERPACKET}$ between transmitting packets on UART_RX. See the UART AC Characteristics section for complete timing specifications. In Mode 4: Transfers are initiated by Eterna asserting UART_TX_RTSn. The UART_TX_CTSn signal may be actively driven by the companion processor when ready to receive a packet or UART_TX_CTSn may be tied low if the companion processor is always ready to receive a packet. After detecting a logic '0' on UART_TX_CTSn Eterna sends the entire packet. Following the transmission of the final byte in the packet Eterna negates UART_TX_RTSn and waits for tTX_INTERPACKET, defined in the UART AC Characteristics section, before asserting UART_TX_RTSn again.		
UART_RX_CTSN	67	O	-	LOW	**UART Receive (CTS) Clear to Send**	DP::M1P1	Not M4 due to M4-2 using the same PC registers and may effect the interrupts
UART_RX	68	I	-	-	**UART Receive**	DP::TX	API UART will be used with the DP acting as master TX and RX lines must 'cross'
UART_TX_RTSN	69	O	-	LOW	**UART Transmit (RTS) Request to Send, Active**	DP::	DP will be using Mode 4.

	Pin #	I/O	Pull	Active	Description	Net	Justification
					Low	M7.GPIO1/INT | LED	Flow control is mandatory to ensure bi-directional transmission. Must be on Interrupt line so that it can be processed in time The DP CTS will notify the Dn when it is ready to accept data. LED to indicate when interrupt is raised
UART_TX_CTSN	70	I	-	LOW	**UART Transmit (CTS) Clear to Send, Active Low**	DP:: M6.GPIO1/INT | LED	Cannot use other M7 pins as we do not yet have control over the DuinoPro library
UART_TX	71	O	-	-	**UART Transmit**	DP::RX	API UART will be used with the DP acting as master TX and RX lines must 'cross'
TIMEn	72	I	-	LOW	**TIME** Strobing the TIMEn input is the most accurate method to acquire the network time maintained by Eterna. Eterna latches the network timestamp with sub-microsecond resolution on the rising edge of the TIMEn signal and produces a packet on the API serial port containing the timing information. This input is always enabled and must be driven or pulled to a valid state to avoid leakage.	3V3	This pin can be neglected as timing information can be determined via API UART connection.
RADIO_INHIBIT	1	I	-	LOW	**Radio Inhibit** This input is always enabled and must be driven or pulled to a valid state to avoid leakage. The RADIO_INHIBIT input enables an external controller to temporarily disable the radio software drivers (for example, to take a sensor reading that	GND	See description

	Pin #	I/O	Pull	Active	Description	Net	Justification
					is susceptible to radio interference). When RADIO_INHIBIT is asserted the software radio drivers will disallow radio operations including clear channel assessment, packet transmits, or packet receipts. If the current timeslot is active when RADIO_INHIBIT is asserted the radio will be diabled after the present operation completes.		
LNA_EN GPIO18	11	O I/O	- -	- -	**External LNA Enable** Control signals generated by the autonomous MAC supporting the integration of an external LNA/PA. See the Eterna Extended Range Reference Design for implementation details. **General Purpose Digital I/O**	N/C	Disabled in Flash
RADIO_TXN GPIO19	13	O I/O	- -	LOW -	**Radio TX Active (External PA Enable/Switch Control)** Control signals generated by the autonomous MAC supporting the integration of an external LNA/PA. See the Eterna Extended Range Reference Design for implementation details. **General Purpose Digital I/O**	3v3	This should already be disabled in Flash. Otherwise pull or tie high.
RADIO_TX GPIO18	12	O I/O	- -	- -	**Radio TX Active (External PA Enable/Switch Control)** Control signals generated by the autonomous MAC supporting the integration of an external LNA/PA. See the Eterna Extended Range Reference Design for implementation details. **General Purpose Digital I/O**	N/C	Disabled in Flash

13.3 Appendix C – Timesheet of CSC

Field	Total Hours
Group	36:35:00
Peter	75:30:00
Aaron	113:35:00
Jamie	139:07:00
Matthew	62:45:00
Yung	78:55:00
Jake	151:59:00
Andy	90:00:00

Net Hours		
Total	748:26:00	hours
	31:11:05	Days