Bi-Directional Communication for Distributed Sensor Network

Alexander Bentley

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work.

Date:

Signed:

Abstract

Distributed Sensor Networks are increasingly being able to have a tangible impact on the day-to-day world because of the growing utilisation of IoT applications and architectures, coupled with improving battery technologies. InTouch Limited own one such sensor network, with the primary purpose of measuring silt levels in gullies to enable better prioritisation of workload for cleaning crews, ultimately preventing flooding. With a re-engineering of the deployed solution, the opportunity is being taken to implement Bi-directional communication into what previously has been a purely mono-directional network. The ability to communicate with deployed devices adds a significant amount of flexibility into the system – something highly sought after when said solution is in place in a multitude of counties across England and Wales, each with specific operational requirements.

Over the course of the following discussion, the prototype devices of the improved sensor network are adapted in order to incorporate bi-directional communication via the use of lightweight IoT protocol *MQTT,* along careful firmware-level changes to the two primary layers of InTouch's network. The solution is developed to the proof-of-concept level, with each stage of communication implemented to the point where commands can be received and enacted by each logical level. The long-term efficacy of the solution is considered, with multiple challenges being identified in relation to the power consumption of the system.

Contents

1. Introduction

1.1 InTouch Limited

InTouch Limited were founded as a two-way radio rental company. Over the past four years they have additionally begun to focus on Internet-of-Things technologies to aid in gully management and flood prevention. InTouch provide probes that can report silt levels in gullies to clients such as local councils, alongside networking infrastructure so that these readings can be collected and monitored remotely. Clients can then see in real time which gullies are becoming blocked by silt, and so can prioritise cleaning crews to visit these areas before they become completely blocked and cause localised or more widespread flooding. InTouch also utilise the vast amount of data collected by performing statistical analysis on a client's gully network to provide them with insights on how the network could be more effectively managed. *SmartWater*, InTouch's primary project for gully management, is now operating on a national scale, with deployments across the north, midlands, and south of England and Wales.

One generation of these devices and their associated software is currently under active deployment, with the next generation in the middle stages of development. This new generation is intended to provide a greater extent of flexibility to the network to enable new modes of data collection and also new configurations of these modes, so as to be useful in a wider selection of real-world scenarios.

1.1.1 Existing Architecture

InTouch's existing deployed architecture (generation two) consists of four primary logical layers: Probes, Data Concentrators ('*DCs*' henceforth), the Data Endpoint, and the Data Processing Platform. Silt sensing probes are typically between 50cm and 2m in length, with a series of light gates at regular spacing throughout. Each probe in this architecture is attached to an ultra-high-frequency (UHF) radio unit, which transmits regular readings taken from the light gates over the air. These transmissions are then received by a nearby DC, which parses the messages and forwards them in batches via GPRS to InTouch's data endpoint. Each DC can have approximately five probes associated with it, spread over an approximate fifty metre radius, before experiencing undocumented behaviour. The data endpoint, hosted within InTouch, then receives all of the messages from the full network of DCs, parses these messages, and splits them into streams that feed into varied methods of further processing, depending on what the incoming information pertains to. For

Figure 1 - Original Probe Head Unit

example, only some areas are enabled to email alerting to clients, so only messages from these geographic areas need to be forwarded to this subsystem.

With the probe units being deployed primarily in roadside gullies or other inaccessible locations, they are universally powered by an onboard battery, intended to keep the probe unit powered and deployed for up to five years.

1.2 Goals

The current generation of *SmartWater* incorporates exclusively one-way communication for the vast majority of purposes. The only exception to this is that the DCs can have their firmware updated over-the-air via *Salt*[1]. However, even in this case, this operation is risky and expensive; a substantial amount of data must be transmitted over-the-air to perform an update, and once initiated such procedures are failure prone, leaving devices in invalid states whilst deployed in the field potentially hundreds of miles away from the office.

This completely removes any possibility of real-time configuration of devices, meaning once deployed they are fixed in their current mode of operation unless a completely new set of firmware is uploaded to them, which is also an unwieldy operation to perform for specific subsets of the network without going through the ardour of defining a new group configuration through Salt. In the situation where a client is interested in gaining a more granular view of a particular probe (maybe they are having some recurring maintenance issues and want to get to the bottom of the problem), there is no mechanism for achieving this post-deployment. If probes or DCs begin to experience operational problems, such as their power source draining more quickly than was anticipated or their effective range beginning to dwindle, it is not possible to tweak their parameters to better account for these conditions whilst a strategy for a fix is implemented.

With this in mind, the goals for the project are as follows:

- Devise and implement a mechanism, or series of mechanisms, by which probe units and DCs can be sent instructions remotely in InTouch's sensor network
- Said mechanism(s) should be lightweight from both a power consumption and data usage standpoint
- Any implemented solution should enable deployed devices to adapt to environmental or operational conditions as they change in real-time

[1] Salt, part of SaltStack: https://www.saltstack.com/

2. Background

2.1.2 Proposed Future Architecture

The generation of InTouch devices / systems currently in development (four) leaves the primary four layers of logical separation intact but introduces a series of improvements to resilience and configurability.

Probe Head – The probe head units are undergoing a redesign from the PCB level upward. Rather than a strong 1:1 coupling between probe unit and head unit, each probe head unit should in future be able to support as the onboard processor can physically listen to, currently theorised to be around five or six due to the power of the onboard processor (*ATMega328PB* – see section 3.1.1).

Data Concentrator – The DCs are also undergoing a near total redesign. The new models will support up to three different types of radio technology (XBee[2], LoRa[1], and RDN's[3] UHF module) interchangeably, and remove the many pitfalls encountered by the previous generation of concentrators (highlighted in 3.1.2).

The proposed lifetime for both the above devices is five years, meaning in normal operation they should be able to be left for a continuous stretch of five years before any engineer needs to visit for so much as a battery change. Again, such a timescale benefits hugely from some kind of communication with said devices because over such a length of time operational requirements will undoubtedly change, even if just subtly.

Data Endpoint – Whilst serving the same purpose, the endpoint has also been re-engineered to increase its maximum serviceable load and the ease with which a new configuration can be added into the processing pipeline downstream of the endpoint.

Data Processing Platform – This platform remains largely the same, besides the continued development of further features.

Outside of InTouch, there is also the addition of some collaboration with *AprilShower* (not their trading name). *AprilShower* own and manage many lighting columns across England. In order to manage these assets, they have implemented an ad-hoc network comprised of a series 'SubMaster' nodes affixed to approximately one in every five lighting columns. InTouch has reached an agreement whereby their probes will be able to send data through this ad-hoc network through to the data endpoint, effectively replacing the InTouch DC with a

[2] XBee Radios: https://www.digi.com/xbee
[3] RDN – RadioDataNetworks: https://www.radio-data-networks.com/

SubMaster node. Whilst having little impact over the system developed as part of this project, the inclusion of this collaboration gives further insight into the overall design ethos of this latest generation architecture: utilising existing infrastructure as much as possible and building systems to reduce to a minimum the number of in-person visits to a deployment site personnel need to perform throughout the lifetime of a product.

2.3 Existing Systems

Distributed sensor networks are certainly not a new concept. As the size to power ratios of computational devices has increased and battery technologies improved, distributed sensor networks have been utilised in such widespread contexts as improving the efficiency within industrial processes[2], monitoring patient health[3], and improving the leisure experience of cycling[4]. Even within the specific application domain of commercial solutions for gully management, as InTouch operates in, there are several competitors on the market already. These include *Kaarbontech*[4], *AMX Solutions*[5], *map16*[6], *Polestar*[7], and *Confirm*[8]. Each of these offer solutions revolving around data collected from sensors embedded in the environment, and at least the solutions from *Confirm* and *Kaarbontech*, if not more, also transmit readings from these sensors across the network in real time, without needing an in-person or drive-by collection[5].

This domain of work also falls under the wider umbrella of the concept of Smart Cities – integrating devices into urban environments in order to improve the efficiency or comfort of urban life[6]. The ability to integrate InTouch's sensor network into a Smart City concept was another of the underlying motivations for carrying out the two-way communication work that is outlined in this paper. The added flexibility and configurability will make future integration into such a paradigm much smoother, as there is not yet a prevailing industry standard for integrating devices into a Smart City network as yet, as noted by Zanella et al[6]. Integrating *Smart Water* devices into a larger framework increases the number of opportunities to utilise the data collected by probes tenfold. Ignoring the potential commercial intricacies of acquiring access to the data, the prospect would provide a common platform by which entities such as the *Department for Transport*[9] or the

[4] Kaarbontech: https://kaarbontech.co.uk/
[5] AMX Solutions: https://www.amxsolutions.co.uk/
[6] map16: https://map16.co.uk/
[7] Polestar: https://polestar-systems.com/
[8] Confirm: https://www.pitneybowes.com/us/location-intelligence/infrastructure-asset-management/eam.html
[9] Department for Transport: https://www.gov.uk/government/organisations/department-for-transport

Environment Agency[10] could gain access to the relevant bits of information which are already being gathered for separate purposes, without having to go to the expense of having their own bespoke sensor network deployed.

The concept of integrating IoT technologies into the Smart City paradigm was further discussed by Lloret et al[7]. with their feasibility study covering the primary utility concerns of a city, how these can be aided by distributed smart meters, and the challenges that need to be overcome to see these in wide deployment. The architecture they proposed has many similarities with that of InTouch's proposed future architecture, with the same three primary logical layers and separations. Protocol families explored by the team included M2M (machine-to-machine) protocols, alongside both bespoke forwarding mechanisms once within the smart meter network and GRPS / GSM once the data transmission needed to reach the data servers at the cloud level. Their highlighted benefits of such an architecture, as InTouch have found, is the increase in productivity that can be attained if a member of staff does not need to visit every individual asset in order to take a reading. However, there is not much discussion of communicating back to the meter devices, primarily because their operation is going to remain the same no matter of any other outside circumstances.

Chan et al.[8] produced a prototype Gully Pot Monitoring system that is highly relevant to the whole of the *Smart Water* architecture. Whilst marginally narrower in application domain (being purely urban environments), many of the challenges encountered by the researchers are ones that also needed to be overcome throughout the development of InTouch's sensor network. For example, the paper notes that several units of their DC equivalent were severely underperforming at the beginning of their field trials. This was found to be due to water ingress damaging the transceiver contained within the housing – something that has also affected InTouch's commercial network and has resulted in the strict usage of IP68 (see section 4) rated enclosures for future deployments. The setup also uses Zigbee radios, of which the XBee modules discussed throughout this paper are a particular form factor of, and so the performance observed by the team can very much be considered analogous to these units. The main points of design at which the two systems diverge are the inclusion of a mesh network between sensor units and the addition of relay devices to account for varying deployment conditions experienced between sensor units and their associated aggregation point (DC equivalent). The mesh network takes advantage of the relative proximity of the sensor units in the stated design – being within an urban environment, each unit is typically within a

[10] Environment Agency: https://www.gov.uk/government/organisations/environment-agency

ten-metre radius, meaning reliable communication between units. With InTouch's network being deployed in more diverse environments, this is simply not something that can be considered reliable to implement with the probe head units this project concerns itself with. It is also likely that implementing such a protocol over the probe head units would drastically increase the overall power consumption of the units beyond acceptable parameters, as they would have to have their radio modules awakened more frequently than the current system provisions for in order to carry messages across units. Whilst the architecture is very similar to that involved in this paper, there is no indication that the devices they have deployed are configurable in any way, and as such they do not state any ways of communicating back to them post-deployment.

A sensor network on a much grander scale is demonstrated by a group of Australian researchers[9]. Their solution involves a different dynamic in that it is their sensor unit (an Unmanned Air Vehicle) that traverses the environment, and it is a set of ground stations that listen for the readings it transmits, whereupon these are transmitted to a set of backend software systems for analysis. This is a completely different paradigm to that currently in operation at InTouch, however this approach may become much more applicable in the near future because of the fact that prospective deployment environments may necessitate for this idea of a roaming sensor (see section 6.1). Regardless, programming the DCs in such a way so that they can cope with a probe head unit changing jurisdiction is likely something that should be carefully considered when implementing the two-way communication aspects. Without consideration of the possibility that a probe head can move around, a DC may encounter an unexpected state where it receives a message intended for a probe that has since been removed, resulting in this message remaining in the DC's queue and becoming stale.

3. System Design

Whilst the bidirectional improvements were originally intended to be implemented across all of the various generations of InTouch devices, this was scoped down before leaving the design phase because of the diminishing returns of implementing the upgrades on older devices. Thus, it was decided that only the latest generation (gen4) would be considered when designing the system proper. This call was made in order to maximise the amount of time that was able to spent implementing a solution onto future hardware – hardware that had been designed from the ground up to incorporate these features. The older devices would have required time consuming workarounds in order to achieve the same level of functionality in an equivalent gen4 device. Additionally, whilst they are still in active deployment, previous generations of devices have no further manufacturing runs planned and there will be getting gradually phased out over next few product cycles where possible. Many of the issues that have led to the phasing out of these devices have been indirectly caused be a lack of two-way communication to the devices themselves, such as the inability to place them into 'test mode' without having physical access to the inside of the casing.

3.1 Hardware

Many of the hardware decisions and devices detailed were made / chosen outside of the scope of the problem of bi-directional communication tackled as part of this project. However, the need for such communication methods certainly influenced a number of elements of design, and the work detailed resulted in various hardware features to be kept for the usefulness or changed for increased suitability in later versions. Hence, much of hardware that was being developed with is outlined below.

3.1.1 Probe Head / Silt Sensors

Figure 2 - Gen4 Probe Head Unit Prototype, with onboard XBee

The current version of the gen4 probe head unit (see figure 2) has had the PCB designed by John Vidler and contains an *ATMega328PB*[10] (referred to as the *ATMega* henceforth) as the onboard processing unit, and an XBee S2CPro radio transceiver for communication. The attached socket can accept input from one or more probe units over serial or SPI (Serial Peripheral Interface) for when there are multiple probes attached. The ability to accept input from multiple probes simultaneously vastly improves the adaptability of sensor deployments – a series of sensors can be deployed without incurring a huge additional material cost of including a separate probe head unit per probe.

The radio module has a vast array of configuration options that can be programmed onto the module itself via Digi's XCTU configuration platform[11]. For example, an XBee module can be configured as purely a transmitter or a receiver depending on the role it is going to fulfil. The primary use of this flexibility in the current system is that of power saving – as many features as possible are disabled on the modules fitted to the probe head units. The sleep functionality is also enabled, allowing the chip to be placed into an extremely low power state via one of the input pins. Before the implementation of two-way communication, all of the XBee's in probe head units were configured purely as transmitters, with them having no need of receiving data.

The *ATMega* is an 8-bit microcontroller designed to allow the user to easily prioritise between performance and power saving when in use on their embedded devices. The capabilities of the *ATMega* have not yet been pushed close to their limits in the course of the implementation of gen4, leaving plenty of room for further features to be added to the probe heads. The oscillator for the unit is configured with a value of 4MHz for the implementation discussed here, but this value could likely be reduced for a deployment scenario to increase power saving.

3.1.2 Data Concentrator

The design of the gen4 DC came with a host of stark improvements over previous iterations. Firstly, and most significantly, the microcontroller backbone of the device was chosen to be a *Particle Electron*[11], a lightweight module consisting of an *STM32F205 ARM Cortex M3* microcontroller, a 3G cellular modem, and 1MB of flash memory. The Electron is part of a series of products developed by Particle intended for use in IoT applications. As such, it is in heavy contrast to the *BeagleBone Black*[12] that the previous generation of DCs used. This unit was far too powerful and cumbersome to be appropriate for the role of a DC, which primarily (entirely for the previous generation) consists of simply retransmitting data after attaching an additional set of metadata. The use of the Electron here cuts the power consumption of the device dramatically, without having any impact of the capability of the device to perform its role. The use of a Particle device also gives access to their

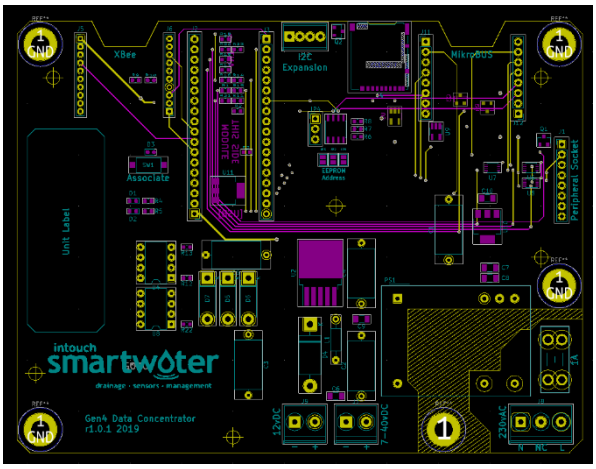

Figure 3 - Gen4 DC Schematic

[11] XCTU - https://www.digi.com/products/iot-platform/xctu

[12] BeagleBone Black: https://beagleboard.org/black

cloud infrastructure, which can be used to manage a fleet of deployed devices, and to even interact with them programmatically, if the functionality has been programmed onto the Particle device itself beforehand. Whilst this platform at first seemed like an out-of-the-box solution to the task of bi-directional communication, there were a number of restrictions that prevented the platform from being pursued further, more discussion on which can be found in 3.3.1.

Beyond the Electron, the new DC has an *Atmel AT24CS01 1-Kbit* EEPROM chip attached[12], accessible to the Particle Electron over I²C. The primary use of this memory module is to utilise the factory programmed unique 128-bit ID number contained within it. This effectively allows for the unique identification of every DC motherboard, no matter how many different radio modules, Electrons or other hardware are swapped in and out over the course of its lifetime. This allows for a convenient addressing scheme of consistently using these ID values when trying to differentiate between concentrators over the network, rather than any programmatically defined values.

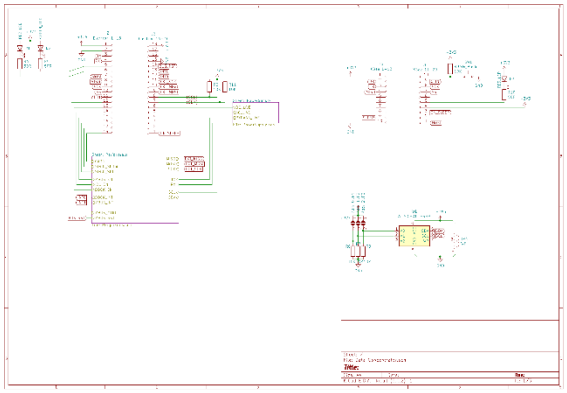

Figure 4 - Circuit Diagram for Gen4 DC assembly

The DC also has two sockets provided, together capable of holding the three different types of radio module that InTouch currently utilises – XBee, LoRa, and RDN UHF. Previous generations of DC had access to only one type of radio module, which struggled in many deployed scenarios and could not be switched out for an improved model because of the bespoke footprint. Having the option of all three types of radio easily was aimed at avoiding these kinds of issues entirely, even if two of the three radio modules proved to be unsuitable further down the line. This reduced the risk as development progressed, meaning that advancements could be made without being 100% sure which kind of radio module would be optimal given X or Y change – they could be changed at a later date if one operated better than another.

3.3 Software

3.3.1 MQTT

Any communication method between InTouch servers and the deployed DCs needed to be lightweight, in order to minimise bandwidth used over GPRS, which is costly per unit of data. It also needed to have the capability to deal with the patchy

network conditions often present between InTouch servers and the deployed DCs, which are typically in less-than-ideal environments.

With the switch to Particle devices as the on-board controller of the DC, the cloud infrastructure provided by Particle themselves was the obvious starting point. The platform is fundamentally REST based, and allows for messages to be sent to and received from Particle devices in the form of events. Events can be published from a device such as an electron to indicate something such as a low-power event. This would then be viewable through the Particle dashboard as an event named 'low-power', potentially with the value of the battery level provided as an additional parameter. The events themselves are simply constructed JSON strings. Whilst straightforward and intended for use with IoT devices, the use of the Particle Cloud for bi-directional communication was vetoed due to the data footprint each communication represented. JSON data is not the most compact, with formats such as MsgPack[13] already improving on the data footprint whilst providing the same freedom of expression. Additionally, utilising the Particle Cloud would add a dependency into the system which InTouch would then not have had control of and would have been reliant on for a swathe of developed features once completely integrated.

MQTT – Message Queue Telemetry Transport[13] - was designed as a lightweight machine-to-machine (M2M) message protocol, intended for distributed, low power devices where network conditions can be unstable. It has recently been accepted as an OASIS standard[13] and seen use in a variety of projects, including ones led by IBM for use with British Sign Language[14]. MQTT implements a publish/subscribe model for participants to send/receive messages. Published messages are sent to an MQTT broker, which then forwards this message to any party that has subscribed to the channel that the message was published to. Channels are described in URL-like strings, such as *dev/<concentrator-addr>/<probe-addr>/control* and participants need to explicitly subscribe to messages on a particular topic, there is no 'join broker' operation which would lead to any default subscriptions taking place. It is possible to subscribe to wildcarded topics, for example *dev/#*, which would subscribe the participant to any topic that had *dev* as a root.

Part of the appeal of MQTT is the fact that it implements differing levels of Quality of Service (QoS) for messages that are published to the broker. The default QoS level simply accepts the published message and forwards this to all subscribed parties. If any participant fails to receive this message, this is not known to the broker and the

[13] MsgPack: https://msgpack.org/index.html
[14] IBM BSL Project: https://www-03.ibm.com/press/us/en/pressrelease/22316.wss

operation of the system continues as normal, and the message that has been sent is discarded. The next level up allows the message to persist at the broker until all parties who have subscribed to that topic have received it. This option is ideal in the context of InTouch's sensor network, as devices are often going to be asleep at the time the command is sent, and the broker needs to be able to keep the message in the system until it has been delivered, rather than the sender having to consistently re-publish the message until it appears something has received it.

MQTT was chosen as the protocol for communication between the InTouch servers and the DCs over the other options available due to the QoS options available in it and its lightweight nature. This also combined with its very low lead time in order to set up and test within the timeframe available. With MQTT becoming a growing standard, the implementation plan for the protocol appeared straightforward, with several pre-existing libraries available for MQTT in the Arduino-like environment on the Particle Electron. This fact removed any implementation challenges that may have resulted in another method, namely the Particle Cloud or the implementation of a bespoke delivery mechanism, being favoured.

3.3.2 Probe Head Firmware

The probe heads are the element of the device network that have the greatest number of constraints placed on them. Of the devices discussed here, only the silt sensors themselves are placed in harsher environments, and this is only because part of the sensor is often submerged in the gully water (something that can also happen, albeit infrequently, to the probe head units). This makes it extremely difficult to visit them to make edits, repairs or replacements, therefore it is of paramount importance that they operate as independently as possible and have the lowest failure rate of any element of the network.

For this reason, designing the operation of the probe head firmware, post alterations, was the most difficult aspect of the design work to find an optimal solution. The design had to incorporate some way of receiving commands without unduly impacting the battery life of the product, or increasing the chance of failures whilst in the field. Options such as polling the InTouch servers upon every upload were considered, as this would not involve the device waking up more often than the current solution. However, such a solution would necessitate the probe head to then wait for a response from the server that may never come. Additionally, it would require the probe head to know which server to contact – it may become difficult if the controlling server were to change address for any reason. Another option was to keep a separate timer running on the probe head that would wake the device at specific intervals in order to receive commands. This was discounted due to the fact that the clocks and timers present on the *ATMega* were not accurate enough over

long durations for this to the feasible – the clock drift would mean that synchrony would quickly be lost with the DC (such synchrony being an expensive thing to achieve in the first place). Even if this could have been overcome, this would still have placed another constant load on the probe head of maintaining the state of this timer rather than fully going to sleep.

The final design that was carried through to implementation involved some of the elements just discussed, but modified or used in such a way so as to mitigate their drawbacks. The proposed solution is as follows:

- Upon every data transmission made by the probe head, it should stay awake in order to listen for a reply for N seconds, where N is dictated by the physical surroundings of the probe (theorised to be an absolute maximum of 180).

- The DC shall be responsible for noticing that a message has been received from a particular probe head unit, and that it is therefore awake and able to receive commands for N seconds. Any commands which are waiting in the queue from the InTouch servers should be sent to the probe head during this time. This will involve the concentrator needing to keep track of the state of the probe head units attached to it, and holding in a queue any commands that are destined for them.

- If no messages are received during the N second interval, the probe head shall go back to sleep as normal

- The DC may send an explicit 'no commands' message, in order to cut short the time which the probe head may have remained awake for otherwise. This could be particularly useful in conditions where N is necessarily large, so anytime where this time can be cut short would be useful for the battery life of the probe head unit.

This operation can be visualised with the following state diagram (figure 5).

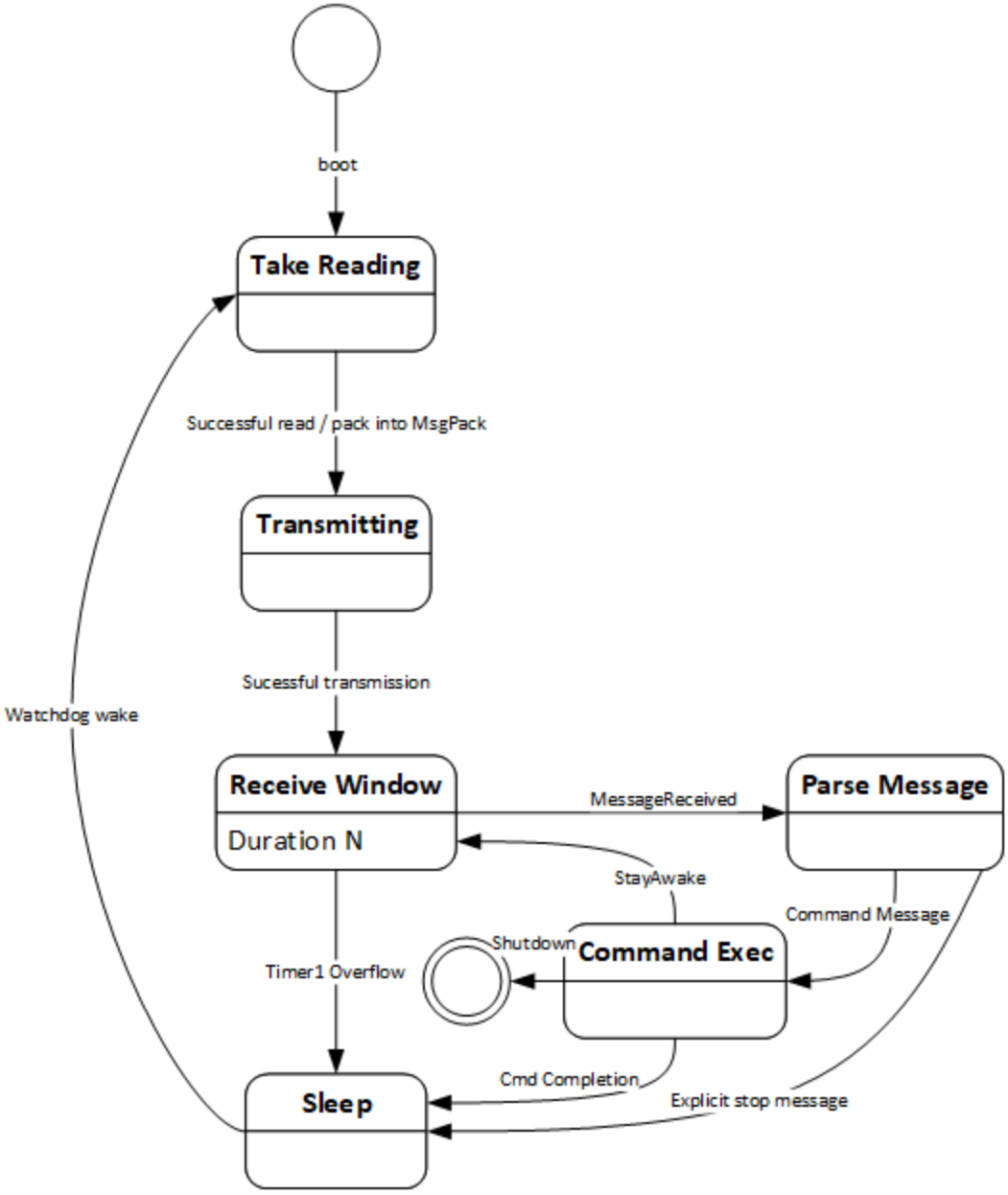

Figure 5 - Proposed probe head firmware state machine

This proposed solution requires three main components to be added to the probe head firmware. Firstly, reception over the XBee radio module would have to be enabled and then programmed into the firmware. A 'stay awake' timer would need to operate over the N second interval in order to inform the controller that the interval had passed, and some way of implementing the instructions contained within the incoming commands needed to be programmed.

The proposed solution is not without drawbacks. It is entirely possible that no messages will ever be received using the described methods – network conditions may never become stable enough for a message to traverse the network in the N

second interval where a probe head device will be listening. Whilst this is unlikely, missed messages (or delayed reception) could cause operational problems, the impact of which is not yet able to be foreseen. However, the simplicity of the approach is in line with the primary design goal of any edits; the power drain of the proposed solution is as minimal as possible when compared to current operation. Another potential issue is the increased load that this method will necessarily place on the DCs – as they will have to maintain a potentially detailed set of state about all of their attached probes / probe head units. With the currently planned capabilities of the latest DCs, this is an additional load that they should be able to cope with. What needs careful consideration in the long is whether or not this additional load will shorten the overall lifespan of the devices as they are left for years on end in deployment.

3.4 Command Definitions

Before beginning development, a set of commands that would already be useful in day-to-day operation of the sensor network were decided upon to provide a baseline and to give an indication of the kind of requirements that commands should be able to fulfil. The end suite of commands was likely to be in excess of these, however they provided a useful reference point and goal.

Await Commands – A simple message, utilising the first QoS level present in MQTT to cause the target device to remain awake longer than usual in order to receive subsequent commands. The target device would need to acknowledge receipt of this message so that an InTouch server knew to dispatch the subsequent commands on a lower QoS level.

Power Save – Enter maximum power saving mode possible, overriding any previous instructions to the contrary. This will likely involve disabling features and increasing the duration of any CPU sleep cycles.

Resume Operation – Target device should resume its default settings (including power, verbosity, etc)

Full Report – Target device is instructed to send InTouch servers a message containing the full set of system state.

Single Report – Target device should send a singular routine message, only immediately, without affecting the ongoing timers for these send events.

Report Rate – This command would be sent with a parameter denoting the desired report rate of the target device.

4. Implementation

4.1 Languages and Environments

The probe head units were programmed in C, with the chip being programmable via a combination of *Atmel Studio*[15] and an *AVR Dragon*[14] programming tool connected to the probe head. The firmware code that had already been implemented at the beginning of the implementation stage was written by Aiden Morton, and was capable of taking readings from silt sensors and packaging these readings into a *MsgPack* suitable for transmission over the air toward the InTouch data endpoint. This operation was also power optimised, with all features of the *ATMega* that were not necessary for this function disabled and power saving sleep cycles enabled and utilised.

The DC firmware code was written in C++ to be suitable for the Particle Electron controller. The Particle Dev IDE[16] was used in conjunction with the Particle CLI[17] for the purposes of flashing the compiled firmware (compiled in the cloud) to the Electron via USB. Whilst it is possible to flash to the devices over the air, USB flashing was used to simply reduce the amount of GPRS data being used over the course of development. Given that the devices were for the most part accessible during development, this caused minimal issues.

4.2 Construction

The DC motherboards first had to be built from their components before development on them could begin. Because many of the components were Surface-Mounted[15] and of a tiny footprint, it was necessary to use a magnifying glass for much of the soldering work. The iron used had the option of an extremely fine nose for these small footprint components and had very granular temperature control (figure 6). This temperature control helped to avoid putting too much heat through sensitive components such as the onboard EEPROM chip. By setting the temperature to relatively low to begin with, whilst also watching the readout for any drops in tip temperature without the solder going into flow (indicating the heat was going somewhere else) meant that no components were damaged in the population process. Again because of the incredibly small size (and therefore low weight) of many of the components, a grippy

Figure 6 - Soldering Iron with fine tip

[15] Atmel Studio 7: https://www.microchip.com/mplab/avr-support/atmel-studio-7
[16] Particle Dev: https://docs.particle.io/tutorials/developer-tools/dev/
[17] Particle CLI: https://docs.particle.io/reference/developer-tools/cli/

mat was utilised so that components were not wasted when a stray breath from the operator saw them scattered far and wide.

Figure 8 - Data Concentrator Motherboard Assembled

Figure 7 – IP68 rated DC enclosure

Once populated, units were placed inside the IP68 (International Protection Marking 68[16]) rated housing (figure 7) that had space for the motherboard itself (figure 8), the Particle Electron battery, and the antennae used by the Electron to connect to the internet over GPRS. The housing being IP68 rated meant that they could cope with being submerged 1.5 metres underwater for up to 30 minutes without any water entering the inside of the enclosure. Whilst the DCs are not likely to ever be fully submerged, such a rating means that they can be deployed in particularly adverse conditions without undue concern over water ingress. The standard also means they are 'dust resistant'[16], again allowing for them to be deployed in the kinds of environments usually surrounding gullies in the UK.

4.3 Probe Head Adaptation

As discussed previously, the probe head units had so far been implemented as purely one-way communication devices – they had no methods of receiving data to the extent that their XBee radio modules had reception disabled entirely. The first stages of implementation involved enabling reception on the radio modules and then beginning to write the necessary code and integrate it into the existing firmware to begin receiving commands. An abstracted diagram of the existing firmware can be found in figure 9.

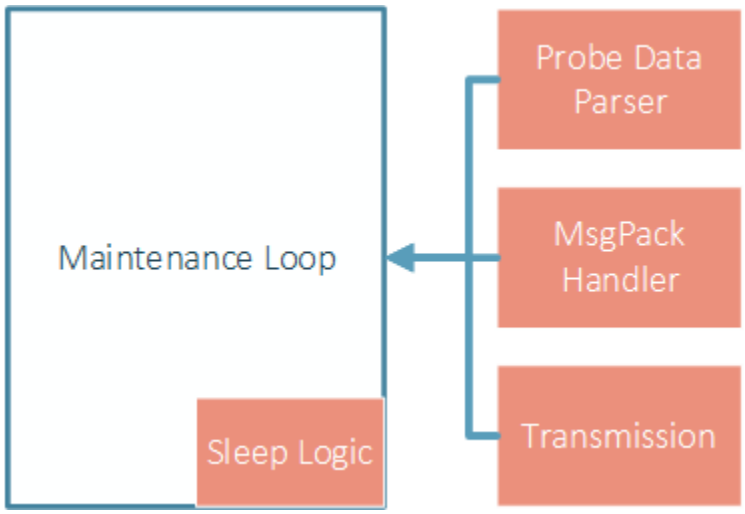

Figure 9 - Probe Head Firmware pre-edit

4.3.1 Radio Receive

The first edit made to the firmware was to enable the receiving of data through the on board XBee radio module, which was connected to the *ATMega* via its Serial1

pin. The firmware already had all of the send functionality fully implemented, and this served as a good starting point to look for the relevant registers needed in the datasheet. With the complexities of receiving incoming bytes over the air handled by the XBee module, the task reduced to the simple requirement of being able to read incoming bytes over the serial line when they were available. With this being a very common thing to want to do, the datasheet was very clear with which registers needed to be edited, and what various values would achieve. Two receive functions, each reading a byte from the serial line in memory, were implemented: one blocking and another non-blocking. The blocking variant would wait a potentially infinite amount of time until there was a byte of data available to be read over the serial line. The non-blocking variant would simply attempt a single time to read and bail if there was no data yet available on that line. Whilst the blocking variant is a more elegant solution and the most intuitive, the non-blocking variant came in very useful later in the implementation when trying to integrate the N second interval in the design (see 4.3.2). In order to test whether the receive functionality was working as expected, another XBee radio module was configured as a transmitter in a Digi development board, whereby series of bytes could then be manually transmitted over the air via the console on *XCTU*. These bytes could then be received by the probe head unit and re-transmitted back out, to be seen re-appearing back on the development XBee's console.

4.3.2 Interval Programming

The next phase of implementation revolved around implementing the N second window where the probe head would be available to receive messages or commands over the radio. For this purpose, the probe head would need some way of telling that N seconds had elapsed, and it should therefore cease listening and go back to sleep. There are three timing modules present on the *ATMega,* each with a suite of prescaler options, and the ability to configure whether they use the internal oscillator as the tick source or an external Pulse-Width-Modulation source. Timer0 and Timer2 are 8-bit timers, which discounted them because of how many overflows of the timer (even with a large prescaler) would have been necessary in order to achieve a timer for the theoretically maximum interval of 180 seconds needed. Timer1 is a 16-bit timer, which when combined with the maximum prescaler value of 1024 and the clock speed of 4MHz, resulted in 31 overflows needed to get an approximation of 180 seconds of time passed. This is where the implementation of a non-blocking receive method was ideal, as it simplified the process of being able to check for the timer overflow flag and enable the controller to notice that the request time had in fact elapsed. Had a blocking receive method been used, the radio receive functions, the timer logic, and the ISR functions would have become very

tightly coupled, as every single receive operation, regardless of context, would have had to perform a check to see whether the controller should, in fact, already have stopped listening for messages coming over the air.

The swathe of registers that need to be configured correctly for the timing modules to operate resulted in a number of misconfigurations even after having consulted the datasheet extensively for the module in question. Shortening the timing window allowed for more rapid testing to see whether the timer had elapsed and disabled reception correctly. This trial-and-error debugging, coupled with extensive debug output being transmitted over the radio back to the *XCTU* terminal, resulted in these issues being tracked down and ironed out without too much impact on the timeliness of the development.

4.3.3 Command Parser / Implementation

The final stage of implementation on the probe head, now that reception of arbitrary data was possible during a specified time window, was to actually make some sense of these bytes and to carry out a set of rudimentary commands. Before doing this, a format for the prototype messages needed to be created. Creating such a format had been a step overlooked during the design phase. The method implemented drew inspiration from the existing over-the-air protocol that existed between the previous generation of DCs and InTouch servers by using the same delimiter (a literal '#') and using the same probe addressing scheme. The format would terminate with a single value, mapping 1:1 with a pre-defined set of commands. This is likely not the solution that would be implemented long term, as it becomes unwieldy to maintain mappings such as these and they are not human readable in the slightest. However, for anything more complex than this, some form of parser would need to be written – something that would have taken more time than available for this stage of the implementation and that would have not taken the device further toward being a successful proof of concept.

With the above scheme in mind, whenever a series of bytes is received over the radio (during the reception interval), the first value the probe head has to locate is the '#' delimiter. It then has to parse all but the last byte and check to see whether this address value matches its own probe head ID, or matches with the address of a probe that is currently attached to this head unit. Once this has been verified, the attached command number can be performed on the probe head / target attached probe. Note that most actions addressed to a particular probe will usually still only have probe head level effects because of the fact that the operation of the probes themselves is largely immutable or otherwise hard coded into the device.

Operations that have so far been carried out remotely include editing the sleep characteristics of the probe head, altering the frequency with which data is transmitted, the listening window, and enabling/disabling features on the *ATMega*, potentially allowing for a module to be disabled by default and then enabled for the specific devices that require it once they are in the field. In additional to the commands themselves, an acknowledge message was also programmed into the probe head, which may in the future allow it to acknowledge a request from the DC to remain awake for longer than usual, or some other out-of-the-ordinary request where the DC would benefit from some form of acknowledgement.

The following diagram displays the same abstracted view of the firmware as previously, only now with the additions and edits visible in purple.

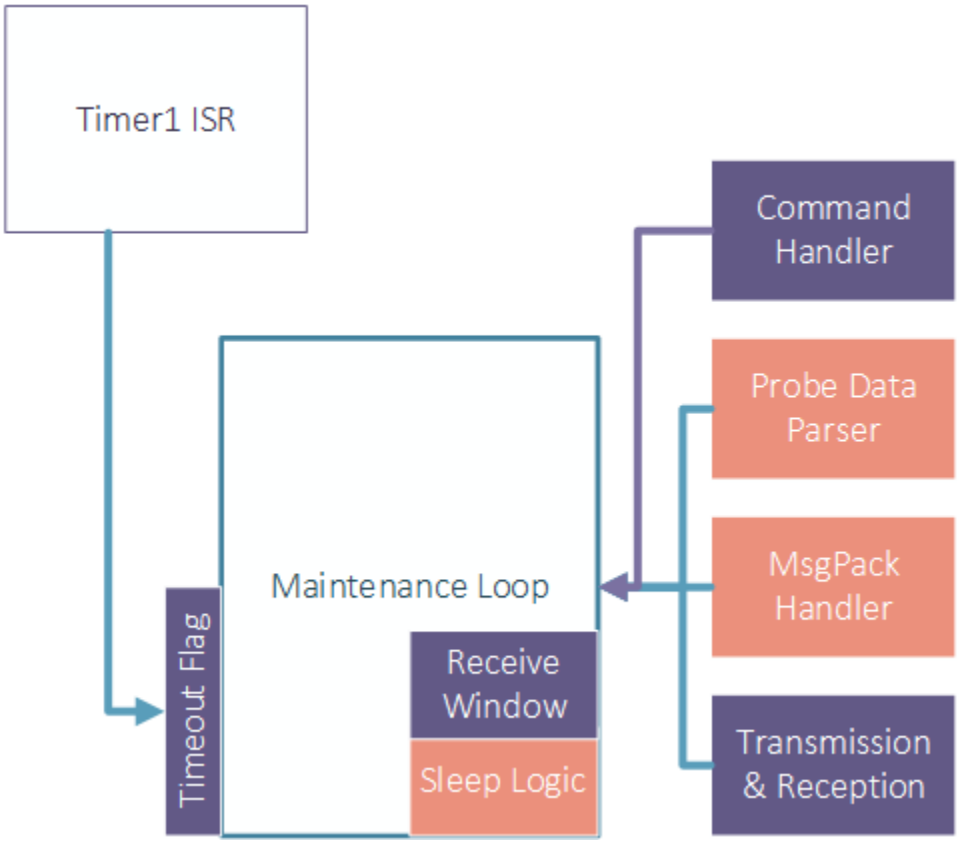

Figure 10 - Probe Head Firmware post-edit, changes highlighted with purple

4.4 Data Concentrator Development

A work-in-progress prototype version of the DC firmware (written for the Particle Electron) was inherited at the start of the implementation stage. However, it did not have any of the features implemented that were necessary for bidirectional communication. What it did have was an overarching structure that helped

immensely with organising the complicated series of classes for the various radio drivers that would eventually be present in the firmware. An abstracted version of this overall structure can be found in figure 11.

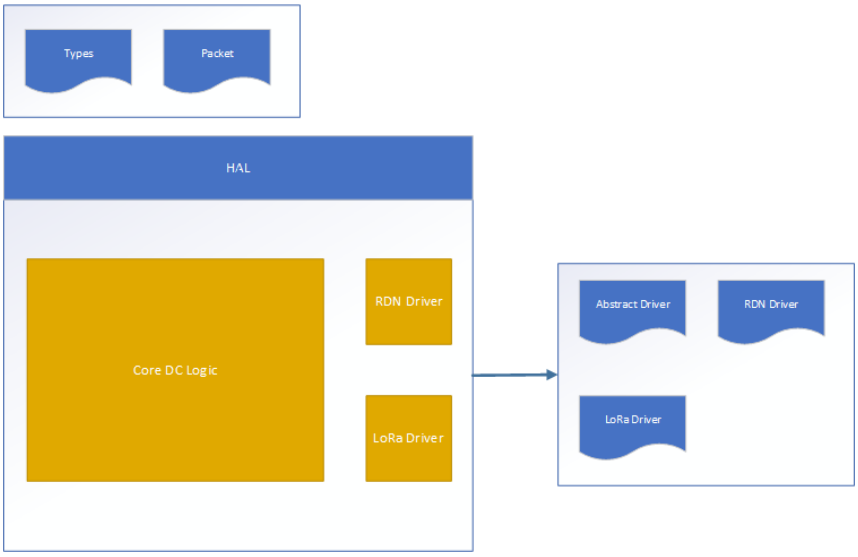

Figure 11 - DC Firmware structure pre-edit

The prototype firmware that had already been written for the DC included drivers for both the LoRa radio and RDN UHF radio modules, but not the XBee modules. The XBee modules were still the ideal ones to be using for this proof of concept work, as they had a proven track record of communicating with each other in other areas of the system, removing one variable from this series of work of the radio modules potentially being incompatible in some way. Similar to the work with the probe head firmware, the first task to complete with regards to the DC firmware was to get this XBee module to both send and receive data. The interfaces for these operations had already been specified by the prototype, so this was a matter of implementing these interfaces. The underlying functionality was much the same as it had been when programming these utilities of the probe head unit, however there was the additional layer of difficulty in the unfamiliar language of C++, and the framework itself also took some familiarisation, but this ultimately resulted in a robust structure that was easy to stick to once learnt.

In order to utilise the EEPROM storage of the *AT24CS01*, code needed to be written in order to identify it over I^2C and then extract the serial number from it that was uniquely assigned at the factory. Performing this operation on boot of the DC would enable it to load its ID value from the EEPROM, in case the Electron had been changed motherboards since the last switch-on. This primarily constituted of reading the datasheet for the *AT24CS01* to see what interaction pattern would yield the hard-coded serial number (the operation was subtly different than when trying to read any other area of memory). The interaction in question required a specific bit pattern to be transmitted over the wire (to the I^2C address of the chip, in this case *0x53*) to indicate that the serial number should be returned over the wire following the next *STOP* signal from the sender. After this value had been found,

the *Wire*[18] library for Arduino was used as a reliable method of actually placing the relevant I²C messages onto the wire and then transmitting the custom bit pattern. Once the serial number had been returned, this was stored in the hardware-abstraction-layer namespace of the firmware, accessible to almost all of the other classes within the firmware.

At this point the communication mechanism to and from upstream servers was ready to be implemented. As per the design, MQTT was the chosen method of message delivery, with an MQTT library already being present within the environment being developed for. A broker owned by InTouch was set up, so that messages could be passed through it going both upstream and downstream. Owning and hosting the broker itself provisions for future security features and other upgrades to be performed at a later date, rather than relying on a third party to provide these upgrades to an MQTT service.

Whenever an MQTT event is received by the device, the *mqtt_event()* function is triggered, whereupon the user must specify how to handle the event, if at all. The only events that will arrive and trigger the function in this fashion are ones that are of a topic that the device has previously subscribed to using *mqtt_subscribe()*. In our case, the DC subscribes to all topics (wildcarded) that begin with either its address, one of its associated probe head units' addresses, or the address of one of those head unit's probes. The rest of the message topic can be used to inform the DC how to handle the message, and the potential parameter accompanying it. For example, */dev/<DC_id>/control/power_save* is very explicitly a control operation (save power) intended for only that particular DC. In this case, no further data is needed to accompany the message (unless varying levels of power saving are at some point implemented), the DC will always perform the same actions when receiving this event.

As well as receiving events, the DC can also publish events back to the MQTT broker, to any desired topic. The current implementation sees the DC publish events to the MQTT broker to indicate that they have received and implemented a particular command that has been sent from upstream – potentially useful to see how regularly messages are being missed by particular DCs in different deployed locations. The upstream and downstream messages are easily differentiated by the topic – for example a message going upstream from a DC with some information about its state would begin */dev/<DC_id>/status/#*, with '#' being any wildcard topic extension. Reading 'status' is enough to inform any listening devices that this is simply a device report from the DC, likely headed to the upstream servers. A DC

[18] Arduino Wire reference: https://www.arduino.cc/en/Reference/Wire

does not have to be subscribed to a topic in order to publish to it, otherwise they would be sent a large number of extraneous messages from other DCs also reporting on their own state.

During development, because the message dispatcher was yet to be implemented (see section 5.1), a dummy MQTT client, *Mosquitto*[17], was used for testing purposes. This simply allowed the user to manually dispatch MQTT messages to a particular topic, and subscribe a terminal to a set of topics, so that they would appear on the console upon being received. In this way, messages could be dispatched to the DCs and their response verified on the corresponding *Mosquitto* client.

The abstracted structure of the firmware codebase after these additions had been made is visible in figure 12.

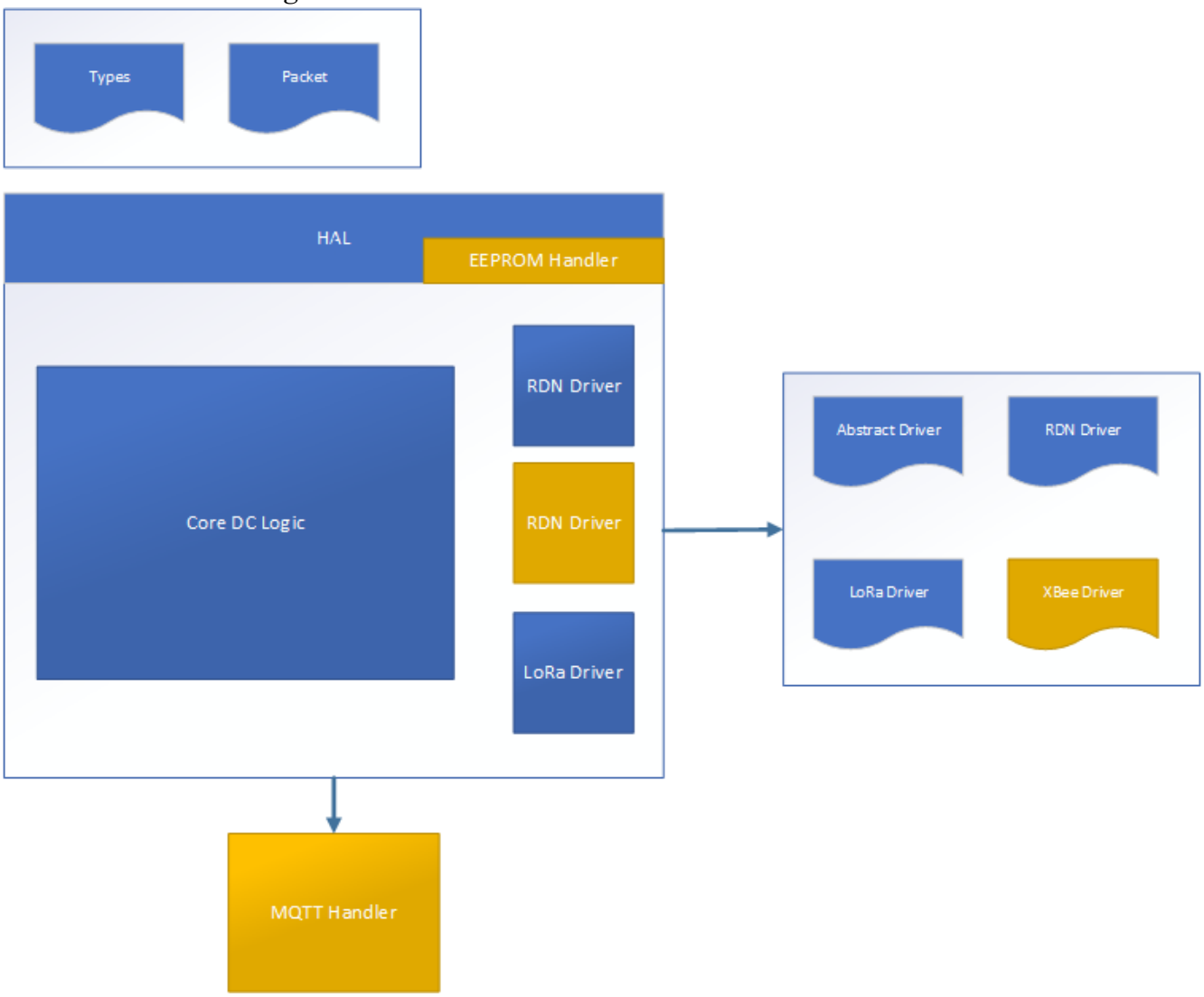

Figure 12 - DC Firmware structure post-edit

4.5 Issues

There were a number of issues encountered over the course of development, the majority of which were overcome. The most significant and consistent challenge presented by the implementation work was the learning curve of working with C++, and to a lesser extent C. C++ contained a raft of primitives, keywords and concepts that I have not encountered previously to this work, having never written a line of C++ before this point. This was overcome via online resources combined with consultation with both Asad Naqvi and John Vidler.

Working with experimental hardware also resulted in its own hurdles. When implementing the driver code to enable the XBee radio module on the DCs to send data to another XBee radio, there was a huge sequence of debugging operations to attempt to find the reason for the solution (sound on paper) was not functioning as expected. With the aid of an oscilloscope and a multimeter, the cause of the issue was happened upon when the XBee module on the concentrator was being held in a certain way in order to take a reading. Bridging PIN_SLEEP and GND on the XBee module with a finger or thumb resulted in the messages successfully being sent and received respectively. This is theorised to be due to a slight misconfiguration of the prototype motherboard which has resulted in PIN_SLEEP being held at a voltage just high enough in order for it to indicate a '1' to the XBee. Unfortunately, the only fix feasible before the delivery of the next board revision was to completely disable sleep on the XBee attached to the DC. This would have impacted on planned concentrator power save testing, however time constraints prevented this from being carried out regardless. Without happening across the cause, this issue could have further impacted time that could have been spent on furthering the implementation, one of the complications of working with hardware that is not yet fully tested.

5. Evaluation

5.1 Further Work

Following is a selection of work elements that fit into the scope of the initial implementation, however had to be cut in some way or dropped entirely due to the time constraints of the process.

MQTT was utilised without any form of encryption for the purposes of this proof-of-concept implementation. Upon deployment, it would be necessary (and good practice) to secure communications between all devices across the network in order to prevent a malicious entity from sabotaging deployed probe units or DCs. MQTT can be performed over SSL, and this is incorporated into their OASIS standard[13]. Whether this would be the ideal approach to securing communication between the InTouch servers and the DCs is unclear, as SSL is a heavyweight protocol and will add significant data footprint to each communication. Because of this, it may well be optimal to instead just encrypt the actual command data and transmit this over MQTT as normal, potentially saving a significant amount of data and achieving a similar level of security. An investigation would have the be performed over the relative merit of these approaches, and others besides, in order to add an element of security to this proof-of-concept work.

Whilst a number of commands were implemented on both the probe head units and DCs, these were far from the levels of fidelity envisioned in the design phase. With the infrastructure for sending and receiving these commands now in place, it would simply be a matter of taking the time to define which parameters should be able to be edited and which behaviours modified via commands and adding these in accordingly – limited logic should have to be implemented at this point further to what has been achieved in this implementation.

Implementation of the central message dispatching service was another element of design that was originally within scope, but ended up being cut due to time limitations. For the purposes of the solution developed, this had an extremely limited impact, as the system could be tested using the *Mosquitto* client mentioned, and functionally the messages from this client appeared identical to the network devices as the ones that would have been sent from the eventual dispatcher would have done. The main difference is simply one of user experience – it is not feasible to craft an MQTT message using *Mosquitto* manually every time that any command is desired to be sent. Some commands benefit from being able to be dispatched in an automated fashion, and a dispatcher would enable non-technical staff to be able to issue the necessary commands without having to be able to craft the packet themselves. Whilst this dispatcher was not implemented within the time allotted,

it would be the least complex piece of the system to implement, particularly because of the way that the current data endpoint has been designed and built (in a previous project with InTouch). The implemented endpoint already makes decisions based on incoming data and forwards messages, with transformed formatting, to a multitude of further processing systems. Integrating a number of automated command responses into this logic would cause no issue, and the system would be able to cope with messages being sent from another source (for example a user) and forwarded back into the network.

5.2 Implemented Solution

The elements of the original design that have been implemented have been implemented to a high standard, successfully dealing with the array of constraints in place for the system. Achieving two-way communication without overly affecting the power consumption of the system was a particularly key requirement; having probe units that can be deployed for five years uninterrupted is a significantly more attractive prospect that those that would need replacing or some other form of maintenance every three. In this regard, the edits made to the probe head firmware to enable reception of commands were very successful. The probe head units consume approximately 30mA under normal operation, peaking at approximately 39mA whilst their radio modules are awake and transmitting silt data. This was measured using a *Keysight N6705 DC Power Analyser*[19] from *Agilent Technologies* (see figure 13). The use of such a high performance power analyser enabled for incredibly high resolution sampling of the power consumption during operation of the probe head units, helping to identify the pieces of code that were causing the largest power drain.

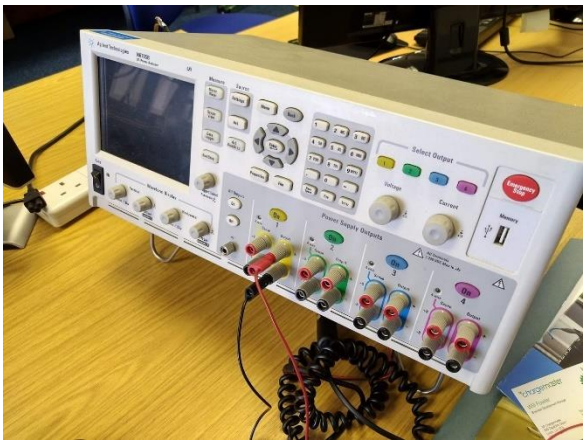

Figure 13 - Power Analyser used during power testing

Having established the power consumption of the probe head units before the reception mode was enabled, the process was repeated, this time with the probe head entering a receive window period of approximately thirty seconds, in addition to normal operation. The power profile remained as before, only once the initial batch of silt data was sent, the probe head consumption did not drop back to 30mA as it had before, instead becoming stable at 35mA. This rose to only 36mA if a

[19] N6705B Datasheet: https://literature.cdn.keysight.com/litweb/pdf/N6705-90001.pdf

message of some type was received during the receive window. The long term impact of such an additional power drain would be entirely dependent on the length of the receive window, and also the frequency with which the probe head unit was configured to transmit silt data. In that case of the receive window being particularly short (around 10 seconds or less), or the silt data transmission period being on the order of hours rather than minutes, the additional drain of receiving commands would likely be accpetable. However, in scenarios where these factors are not the case, the additional power drain would begin to take a toll on the lifetime of the product. Whilst the peak power consumption of reception is less than that of transmission, the transmission power peak is extremely brief, and lasts only as long as it takes to transmit the message over the XBee radio. Considering this takes a maximum of two seconds, even a short receive window will begin to consume three times as much power as a transmission event in a good scenario.

These findings place an emphasis on the importance of sending explicit 'stop listening' commands from the associated DC as early as possible, and ensuring that the receive window itself is configured optimistically – missing occasional messages is far less damaging than continual power drain resulting in a reduced product life expectancy. In cases where the receive window must be configured to be a large amount of time, or where the required silt data report rate is higher than typical, it may be necessary in future implementations to configure the receive window to only occur once every N transmission cycles. To account for this, either the DC firmware would have to be made so it could account for this, meaning a larger amount of state being stored within its memory, or a combination of edits could be made so that a 'receive window' flag be sent with any transmission made by the probe head, indicating whether it would remain open to reception after finishing transmission of the attached message. This was actually an approach considered very early on in the design phase, however was discounted in an effort to minimise the amount of load being placed on the probe head units, in terms of memory, computation, and the actual code footprint of any firmware.

The overall code quality of the implemented features has been maintained through regular code review and careful version control using git. The vast majority of the code written into the system for the described features can be integrated into the production product without changes – the only exception being the current command parser (or non-parser) present on the probe head units, which would have to be replaced with a full version to avoid incurring a tech debt at a later date where this becomes entrenched into the design.

6. Conclusions

The benefits of having a system with bidirectional communication enabled became clearly apparent throughout the implementation and design process contained within this project. Even during requirements gathering with InTouch, the list of potential applications of such a capability came readily to all those with whom it was discussed, along with a similar length list of all the headaches it would have saved them. With reference to the original aims of the work:

"Devise and implement a mechanism, or series of mechanisms, by which probe units and DCs can be sent instructions remotely in InTouch's sensor network"

A series of mechanisms were indeed implemented, including both directions of Probe Head – DC communication, and the DC side of the Data Endpoint – DC communication. Whilst the Data Endpoint is not yet equipped to dispatch commands across the network, all of the groundwork is present in order for this to happen in short order with a little more development work. This goal can be considered met within the scope of this project.

"Said mechanism(s) should be lightweight from both a power consumption and data usage standpoint"

Measures were taken to address both of these requirements. Bandwidth usage has been successfully kept to a minimum through the use of lightweight protocols such as MQTT and being realistic with the amount of delivery reliability that can be expected from the probe head units in order to reduce the total number of messages in any exchange. Further data should still be obtained on the total number of bytes that are exchanged across the network as part of the two-way messaging functionality, as this cannot currently be verified past the fact that the design and theory behind all of the implemented mechanisms have kept data usage to an absolute minimum, and the SIM cards on the DCs have so far not run into any issues with data caps.

Power usage is the area that may need improvement, or a subtly different approach implemented on the side of the probe heads, in order for the solution to see long term success. Whilst the power consumption of the newly implemented features remained proportional to the existing consumption requirements, the fact that the features require to be running for longer than the other operations of the probe head, the battery life of the product appears as though it would suffer under the current strategy.

"Any implemented solution should enable deployed devices to adapt to environmental or operational conditions as they change in real-time"

The implemented solution does indeed allow for any number of commands to be implemented on the devices, with many of those discussed in the design phase already functioning in a proof-of-concept manner, such as changing the sleep behaviour of a unit or request more frequent updates from it. The reaction time of the network, as a whole, would still be on the order of hours (due to the device sleep schedules) rather than minutes or seconds. However, with the sensor network being concerned primarily with silt build-up over time (rather than any instantaneous events), this still constitutes as 'real-time' for the purposes of the current application area.

Overall, the work carried out as part of the project was largely successful. A proof-of-concept level prototype was produced that contained the core functionality required in order to facilitate bi-directional communication between devices. Whilst there is plenty more refinement and improvement to be done, the system has definitely moved in the right direction and closer to being the more flexible and configurable successor to generation 2 that it was designed to be.

6.1 Future Work

None of the discussed work considered the new partnership with *AprilShower* that has developed at InTouch. More work would need to be conducted in coordination with *AprilShower* themselves in order to pass data back through their network to a variable end location as well as pass data upstream from the DCs towards a single endpoint. There have been brief discussions with them thus far, and MQTT has been discussed as an option for implementing bidirectional communication through this medium also. It is unclear how much work would need to be done on either side, mainly because of the black-box nature of *AprilShower*'s network at the present time. The current system is to simply attach a magic number to any InTouch traffic, and *AprilShower* ensure that it is forwarded toward the InTouch endpoint. Having observed the company reconfigure a portion of their SubMaster nodes ready for a demo operation in the south of England, it is clear that *AprilShower* themselves have some method of communicating back with their SubMaster nodes, which has the potential to work for this same purpose with some minor tweaks.

Whilst MQTT proved to be effective for the application area of InTouch's network, there exists a version of MQTT that is potentially even more relevant and optimised

for such a use case: MQTT-SN – MQTT for Sensor Networks[18]. Investigation of whether this option would be viable would certainly be worthwhile, considering InTouch's network is indeed a sensor network and Zigbee technologies, in the form of the XBee radios, are being integrated into it further as the current generation is developed.

Recent partnership opportunities currently under discussion at InTouch could see another mode in which their devices operate, as mentioned earlier in section 2.3. This is due to the prospective partners having assets that they wish to be monitored in incredibly hard-to-reach locations, both physically and legally (such as the railway). This would place even further requirements on the resilience of any device deployed. Additionally, it would mean a strategy for gathering data and sending commands to and from devices would have to be implemented that accounts successfully for the fact that the receiving end of the communication would only be present for seconds through the course of day. In the case of assets being monitored for the railway, this would be the DC being driven over the probe head units whilst on board a train, meaning that not only is the device only present in the vicinity for a very short amount of time, it is travelling at great speed and with a substantial amount of radio-shielding metal in the way at the time is it present.

6.2 Learning Outcomes

An entire language was learnt for the purpose of this implementation – C++. This was simultaneously the most challenging aspect of the work and yet the element that is most likely to be useful in future projects. Working with InTouch's prototype hardware further increased the level of familiarity and confidence felt when working with embedded systems, both in a hardware and software capacity. Surface-mount soldering was a completely new experience, but highly satisfying when performed correctly. Exposure to incredibly high-end hardware such as the DC Power Analyser we had access to provided a very unique insight into analysis methods and techniques that a hardware manufacturer should go through before releasing any productionised system. In summary, I feel far more comfortable working with embedded systems at all stages of development after completing this project than I ever would have beforehand.

Should the project be carried out again, or indeed continued, greater care would be taken with the time scaling and scoping of the system to be implemented, especially in areas where completely new technologies had to be learned in order to be implemented. In this instance, the impact of not being able to complete these aspects was minimised by the fact that the work was going to be continued past the conclusion of this project by InTouch regardless. However, in another context, a

significant amount of insight may be lost through features having to be rushed to implementation when in fact it would have been much more prudent to focus the time properly on other areas of the work and foregone some features entirely.

6.3 Acknowledgement

None of the work completed over the course of this project would have been possible without the use of both InTouch Limited's equipment and expertise. Particular thanks are owed to John Vidler (Tech Lead InTouch Ltd & Lancaster University), Dr. Asad Naqvi (Lancaster University), Aiden Morton (Lancaster University), and James Cheese (Tech Lead InTouch Ltd) for technical assistance, both in troubleshooting and simply learning the many new technologies encountered over the course of the implementation. Much of the pre-existing code inherited at the start of the project was of authored by one or more of the above. Neither myself nor Aiden would have been able to complete our respective project ideas had we not been invited back to InTouch to do so by John Walden, Managing Director InTouch Ltd.

References

[1] LoRa Alliance Technical Committee. (2017). *LoRaWAN 1.1 Specification*. LoRaAlliance. Version 1.1. Accessed May 2019. https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf

[2] Zheng, T., Gidlund, M., Åkerberg, J. (2016). *WirArb: A New MAC Protocol for Time Critical Wireless Sensor Network Applications*. IEEE Sensors Journal. Volume 16. Issue 7.

[3] Gope, P., Hwang, T. (2016). *BSN-Care: A Secure IoT-Based Modern Healthcare System Using Body Sensor Network*. IEEE Sensors Journal. Volume 16. Issue 5.

[4] Eisenman, S., Miluzzo, E., Lane, D., Peterson, R., Ahn, G., Campbell, A. (2007). *BikeNet: A Mobile Sensing System for Cyclist Experience Mapping*. ACM Conference on Embedded Networked Senssor Systems (SenSys).

[5] Ibbitson, A. *SmartWater Market Insight Report*. AD Ibbitson Consulting and Interim Ltd. March 2019 – *available upon request*

[6] Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, Michele. (2014). *Internet of Things for Smart Cities*. IEEE Internet of Things Journal. Volume 1. Issue 1.

[7] Lloret, J., Tomas, J., Canovas, A., Parra, L. (2016). *An Integrated IoT Architecture for Smart Metering*. IEEE Communications Magazine. Volume 54. Issue 12.

[8] See, C., Horoshenkov, K. Abd-Alhameed, R., Hu, Y., Tait, S. (2012). *A Low Power Wireless Sensor Network for Gully Pot Monitoring in Urban Catchments*. IEEE Sensors Journal. May.

[9] Malaver, A., Motta, N., Corke, P., Gonzalez, F. (2015). *Development and Integration of a Solar Powered Unmanned Aerial Vehicle and a Wireless Sensor Network to Monitor Greenhouse Gases*. Sensors 2015. Issue 15. 4072-4096

[10] Microchip. (2018). *AVR Microcontroller with Core Independent Peripherals and PicoPower Technology*. Accessed April 2019. http://ww1.microchip.com/downloads/en/DeviceDoc/40001906C.pdf

[11] Particle. (2018). *Cellular connected Internet of Things development kit with a small footprint powered by a powerful STM32 ARM Cortex M3*

microcontroller and a uBlox cellular chip. Version 5. Accessed April 2019. https://docs.particle.io/assets/pdfs/datasheets/electron-datasheet.pdf

[12] Atmel. (2015). *AT24CS01 and AT24CS02 I2C-Compatible (2-wire) Serial EEPROM with a Unique, Factory Programmed 128-bit Serial Number*. Accessed May 2019. http://ww1.microchip.com/downloads/en/devicedoc/Atmel-8815-SEEPROM-AT24CS01-02-Datasheet.pdf

[13] OASIS Message Queueing Telemetry Transport (MQTT) Technical Committee. (2014). *MQTT Version 3.1.1, OASIS Standard*. Accessed May 2019. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718019

[14] Atmel. (2016). *The Atmel AVR Dragon Debugger*. Accessed May 2019. http://ww1.microchip.com/downloads/en/devicedoc/atmel-42723-avr-dragon_userguide.pdf

[15] Surface Mount Council. (1999). *Status of the Technology, Industry Activities, and Action Plan*. Accessed May 2019. https://web.archive.org/web/20151228182745/http://www.ipc.org/4.0_Knowledge/4.1_Standards/smcstatus.pdf

[16] International Electrotechnical Commission. (2013). *Degrees of protection provided by enclosures (IP Code)*. Edition 2.2.

[17] Light, R. (2017). *Mosquitto: server and client implementation of the MQTT protocol*. The Journal of Open Source Software. Volume 2. Number 13.

[18] Stanford-Clark, A., Truong, H. (2013). *MQTT for Sensor Networks (MQTT-SN) Protocol Specificaiton*. Accessed June 2019. http://www.mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf