Implementing NB-IoT: Communication with a Load Cell

Johannes Almroth

15th December 2019

Abstract

IoT technology has been proclaimed as a new technological prowess that will change our economy as well as our cities and way of living. Despite these bold statements, IoT is far from being easily implemented by companies not directly working with any of the enabling technologies, such as telecom. Narrowband-IoT (NB-IoT), a new radio protocol focusing on wide area coverage and low power consumption, is being heralded by the 3GPP as one of the key technologies necessary to push society into the age of IoT. NB-IoT networks are still extremely new in a lot of countries, and while the SIM-cards necessary to use these networks can be readily purchased from telecom companies, the lack of implemented projects might scare the everyday layman looking to implementing IoT within his/her business. The purpose of this paper is to provide an example for how an IoT device can be implemented in practicality, specifically with a scale. A micro-controller is hooked up to a load cell, from which the data produced is sent to the net via a cloud platform.

Contents

List of Figures

Chapter 1

Introduction

IoT (Internet of Things) is a broad, diverse and growing field within the IT sector. Many organizations predict that it will come to impact large areas of our daily life, and many telecom companies are experimenting with different kind of real world applications that can benefit from this change. The basic idea is the same for everything, which is to take a device and connect it to the internet. Examples range from simple toasters to complex self-driving cars.[**?**] The focus is to enable communication between devices without the need for a human middleman, thus optimizing whatever application is being implemented. A simple example is a building equipped with multiple IoT-enabled thermostats, which are controlled by a central heating system. Given effective software, heating can be regulated in an energy efficient manner while still keeping visitors adequately warm throughout the day. Another example might be a parking meter, which can forward the availability of its parking spot to some central system which in turn forwards the closest available spot to an end-user. The potential applications are numerous, but factors such as energy consumption and security have proven huge roadblocks that pose huge challenges to most IoT projects.

Vetek is a Swedish scale supplier located in Väddö, situated approx. 100 kilometers north of Stockholm. Vetek constructs their own scales and weighing systems, as well as reselling products from other manufacturers.[**?**]

Vetek aims to improve their services, and as such are interested in the possible use cases of IoT technology, and ultimately see how that can be applied to their own products. With something as simple as an IoT-enabled scale, they can offer customers products that can be placed in remote areas without needing constant checkups, enabling long term monitoring and easier analyzation of data. An example might be monitoring road salt depots, to enable smarter refill routes during winter time, or a fodder station to map the behaviors of local wildlife.

The biggest challenges for this type of device lies in energy consumption and broadcast range. Low energy consumption is needed so that any maintainer doesn't need to make constant check-ups to switch batteries all the time. This poses limitation on the type of scale that can be used, which in extension affects parameters such as scale accuracy and capacity. A wide broadcast range is needed so that the device isn't limited by having to be close to a base station. This puts restraints on what type of communication protocols can be used, as traditional ones such as Wi-Fi and Bluetooth won't work in the aforementioned examples.

With the advent of IoT, the 3GPP (a standardization organization for telecom) has developed new wireless communication protocols intended to be used by these

devices. One of these, the NB-IoT (Narrowband-IoT) protocol is particularly suitable for the challenges mentioned above, as its focus lies (among else) in wide area coverage and long battery life. A microcontroller (a small computer) is needed to handle the data polled from the scale, as well as sending it via some wireless communication protocol. The microcontroller chosen for this project is a FiPy, as it has the capability to handle multiple wireless technologies, one of them being NB-IoT. The only other technology needed is some form of power source, as well as an ADC (Analog-to-Digital Converter) that connects the microcontroller and the scale.

In this paper, an attempt is made to implement the above. Due to hardware difficulties, a functioning connection between the scale and the microcontroller couldn't be established, so a virtual scale was implemented instead. Some behavioral restrictions are placed on the code running the microcontroller, to closer resemble a real-world application. Examples of these behaviors are handling of erroneous data and disconnection of the scale.

Chapter 2

Background

2.1 General Background

Internet of Things (IoT) has been lauded as a world-changing technology that will significantly affect our economy as well as our way of living. In a report by the GSMA, the total number of IoT devices is estimated to triple by 2025, bringing the total number of devices to $25.2 *billion*. Meanwhile, the global IoT revenue will fourfold from 2018, increasing it to $4.4 *billion*.[**?**] While there undeniably is a lot of excitement and potential economic impact associated with IoT, currently, many consumers just associate the term with connecting a common toaster or coffee machine to a Wi-Fi network. While this technically fits the definition for an IoT device[**?**], the significant use cases will probably be implemented with different sensors, such as scales, thermometers, etc. that will further improve automatization & optimization processes. As an example, the key categories within the predicted growth is smart homes (e.g. security devices) and smart buildings (e.g. energy consumption sensors).[**?**] For the predicted growth to happen, businesses need to take a chance and work on projects that implements different IoT technologies, and to enable this, the GSMA has developed some LPWAN (Low-Power Wide-Area Network) protocols that focus on different key aspects that make IoT possible. Some of these aspect include long battery life, high connection density, indoor coverage and geo-tracking capabilities. Aside from security, one of the biggest challenges regarding IoT devices relate to limitations arising from energy infrastructure. As mentioned earlier, one of the core issues NB-IoT aims to achieve is to be a low-power technology, thus decreasing the maintenance needed for battery-powered devices. A claim often paraded with NB-IoT is that it enables a battery-time of up to 10-years[**?**], though it's worth mentioning that over such a period of time the underlying IoT technology (in the form of microcontrollers/sensors) will probably require more frequent maintenance than the batteries themselves.

2.2 NB-IoT vs. LTE-M

The two most prominent of the protocols developed by the 3GPP are NB-IoT and LTE-M. LTE-M has the most functionality, including voice capabilities and device positioning. Thanks to its wider bandwidth frequency it also has lower latency and boasts a data rate up to 1 Mbps.[**?**] In return, device complexity and costs are higher compared to NB-IoT. The focus of NB-IoT was to enable indoor coverage, low cost development, long battery life and high connection density, which makes the

technology ideally suited for low data rate applications in extremely challenging radio conditions.

2.3 NB-IoT Market in Sweden

According to Swedish telecom company Telia, they were the first to introduce the NB-IoT technology in Sweden, as well as the Nordic countries overall.[**?**] They further claim that their network will be in range for over 99.9% of Sweden's population, as well as provide a speed of 200 kb/s in more than 95% of the country.[**?**] The grand opening of the network was on the 24th of May, and pilot projects were conducted as early as a year before this, in multiple locations across the country. Telia currently offers a starter kit for any actor interested in the technology, with a trial period of 6 months that includes access to Telia's IoT portal and APIs as well as 5 SIM cards, each with a 30MB data cap per month. Telia doesn't seem have many competitors when it comes to the Swedish IoT market, though Tele2 have partnered up with Nokia to offer similar services, and according to a press release from 2018, they have rolled out both LTE-M and NB-IoT across their networks.[**?**] Telenor has launched a IoT network in Norway with NB-IoT functionality in 2018[**?**], and according to an exchange with their customer support, followed suite in Sweden in the beginning of October. **TODO: How do I reference a private email conversation?** The fact that Telia already has partnered up with a multitude of cities and companies give the indication that they have a head start in the market.

Chapter 3

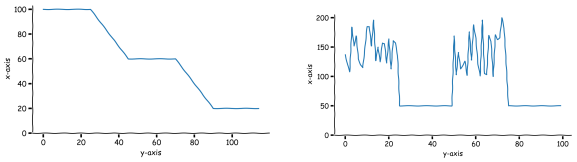
Requirements

In this section we will discuss some the requirements imposed on the IoT-enabled scale device that will be implemented, and what behavior we wish to see from the device in the case of different scenarios. These areas are: data reading rate, data transmission rate, sensor failure as well as sensor disconnect. These areas were chosen because of their relative simple nature, as well as their relevance to other similar devices.

3.1 Data Reading Rate

In most IoT devices the relevant data is provided by some form of sensor, whether it be a scale, thermometer or something entirely different.[**?**] The type of sensor being used has a huge impact on the IoT device, especially when considering that they have to be powered by the same energy source. The simplest way of deciding when to poll data from a sensor is to let it do so at a fixed and constant rate, often enough to be relevant, and seldom enough as to not waste precious energy. However, if within the context of the application we can conclude that no data needs to be polled (for a while), then subsequently no data will need to be sent, and thus we save energy on both ends of the system. For some applications there might even be longer periods of downtime where it's not relevant to conduct monitoring on the given sensor, *e.g.*,during nighttime, closing hours, etc. Another interesting angle is modifying the reading rate depending on the data itself. A simple example of a behavior would be to have a slower reading rate at stable values, and increase it when experiencing large enough changes. Given the conditions of an IoT device powered by batteries, it's not unreasonable to assume that readings might not always be accurate at times. Depending on the sensor, spikes and drops of false values might occur, and not taking these scenarios into account would be prudent.

In this paper, we want the device to change its reading rate depending on the data values, such that the rate increases at periods of activity and change, and lower the rate when the data stabilizes around a given value. We also want the device to try to take false data values into account.



(a) Decreasing values (b) Graph with spikes

In figure **??** we see the sensor outputting stable data around a fixed value, to then experiencing a decrease, two times. We want the readings to increase during the data changes, and decrease during the stable plateaus.

In figure **??** we see the sensor outputting some data values with sudden spikes in them. For the sake of simplicity, we will assume that our scale has the maximum capacity of 100 kg, and thus can easily discern that any value above this is a false reading. In other contexts, false readings might be harder to detect and handle.

3.2 Data Transmission Rate

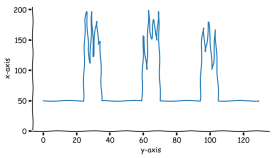
The transmission of the data is in most cases the operation that consumes the most energy for a lot of devices. The energy amount depends on the hardware and protocol being used, which in turn may affect if a device is programmed to send updates once every minute, or once every day.

The requirements set on our device states that we need to send the current value of the scale once every ten seconds.

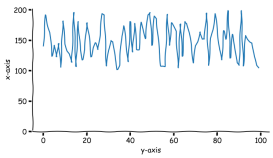
3.3 Sensor Failure

In this paper, we define sensor failure as a sensor giving too many unreliable or false data values to be considered functional. The goal of identifying such a state in an IoT device is to prevent unstable data from being interpreted as valid, which in turn can save the end user from unwanted consequences. Depending on the longevity and purpose of the device, the threshold of when to declare a sensor as failing may differ, especially as this state can be quite fluid. A functional sensor means different things for different devices and applications. A simple way might be to conclude that if x% of data is considered invalid during the last 24hrs, an alarm should be raised to the device administrator. Complications arise when failures need to be reported quickly, or estimated more thoroughly. It's also possible that the sensor can be temporarily unreliable due to external circumstances, and given enough time, these circumstances might pass. On one extreme you can have a device that reports failures too frequently and bogs down whatever dashboard is handling it's status report. On the other, you can have a device taking too long to determine a sensor failure that otherwise useful data could have been monitored if an error had been raised in time.

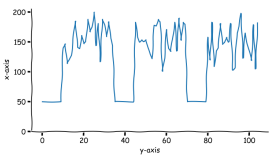
The requirements set on our device state that it has to have some form of self-regulation of when to send these error signals, allowing the sensor some time to recover. If the sensor doesn't recover within a given timeframe, or outputs too much erroneous data, it will raise an error. The reason for still raising an error even though the device recovered is that the sensor might be in need of maintenance if the % of invalid data is too high. This all depends on the filtering of invalid data, as well as at what data threshold the device loses effectivity.



(a) Sensor recovers after some invalid reads



(b) Sensor doesn't recover after invalid reads



(c) Sensor produces too much invalid data

In figure **??** the sensor outputs some invalid data, yet it recovers. No errors should be raised.

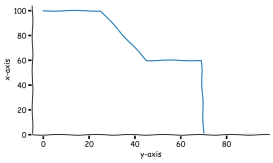
In figure **??** the sensor doesn't recover from reading invalid data, and an error should be raised.

In figure **??** the sensor does recover, but still outputs enough invalid data that an error should be raised.

3.3.1 Sensor Disconnect

We define a sensor disconnect as when no credible data is being produced at all. If the sensor doesn't recover, immediate maintenance is needed for any continued functionality. In this device we know that a sensor disconnect results in the value 0.0 being returned at every poll by the microcontroller to the sensor. While this might be a valid read in some scenarios, in a real world application we would probably never get such a stable value at exactly 0.0. Rather, it would fuzz around at maybe 1.5$\tilde{0}$.0. Disregarding this, we can also know that sensor has disconnect if the data values drop to 0.0 at a too quick pace to be a real-world measurement.

The requirements on this device states that it should raise a disconnect error if the sensor doesn't recover within a given timeframe.



(a) The sensor disconnects

In figure **??** the data values drop from 60 to 0 in the span of one data point, which would indicate a sensor disconnect.

Chapter 4

Design & Implementation

The work in this paper is divided up into two major areas; the hardware and the software implementations. The aim of this section is to give an overview of the implementation and the design decisions made for each area.

4.1 Hardware Implementation

A similar paper conducted at KTH earlier this year served as the main baseline for how the hardware was to be setup.[**?**] The main components are the microcontroller, the scale as well as the ADC (Analog-to-Digital Converter). Apart from this, an adequate power source is needed to provide energy to all components.

4.1.1 Scale

The scale used for this paper is a Tedea Huntleigh - Model 1022. It's a small and simple model, and the specific device used in this paper had a maximum capacity of $\tilde{5}0$ kg.[**?**] In figure **??** we can see the labels of the four wires needed to hook up the load cell. The *Input+* and *Input-* signify the voltage input and ground. [**?**] *Output+* and *Output-* will output a positive respectively a negative charge of $\tilde{1}.5$ voltage . During weighing, the internal resistance in the load cell will change ever so slightly, and the two outputs will have a small difference in the millivoltage range. This difference represents the weight measurement, and can be translated to a corresponding kg/lb value. In general, the more voltage the scale is supplied with, the greater this millivoltage range can be, which (in theory) means larger accuracy when weighing. Bear in mind that no two load cells are the same, and need to be calibrated to output the correct kg/lb value. Even though no calibrations were made to the load cell during this paper, an increase in voltage did correlate to an increased stability in the values produced.

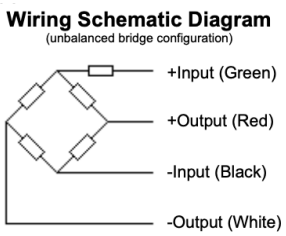


Figure 4.1: The wiring schematic for the load cell

4.1.2 ADC

To convert the millivoltage output from the load cell into a digital signal, an ADC is needed. The device used in this paper is an HX711, and apart from being a converter, it also serves as an amplifier for the load cell signal. We can see the front of the piece in figure **??**, and on the left side are the pinout where all the wires from the load cell should be connected. It's worth bearing in mind that the color coding of the wiring is not the same for all load cells, and the backside should be checked so that the connections made follow the correct wiring schematic.

It outputs data via two of its pins, the DAT and the CLK. The CLK pin will output either 1 or 0 depending on if the HX711 is ready to send data. When it is ready, the DAT pin will send a series of 0s and 1s that can be converted from binary to a decimal value, which will then represent the output of the load scale.[**?**] Multiple code libraries have been written to handle this for the user, and the only thing left to implement is to specify which pins are being used by CLK and DAT respectively. For this paper, a library written for micropython was used.[**?**]

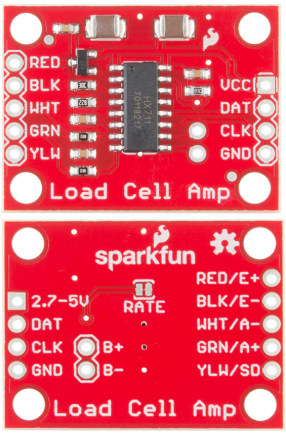


Figure 4.2: The front and back of the HX711

4.1.3 Microcontroller

The microcontroller used for this paper was the FiPy development board from PyCom. It boasts a wide range of capabilities when it comes to communication protocols, NB-IoT being one of the five available.[**?**] With the supplied expansion board, connections via pinout is possible. It runs on micropython, which is an implementation of Python 3 optimized to run on microcontrollers.[**?**]

4.1.4 Power Source

In the early stages of the project, the hardware was powered via USB cable from a computer. Since the USB was of type micro, the voltage output was at 5V. The benefit of this setup was a simple wiring schema where each component powered the next in line. The drawback was that the FiPy could only supply the ADC with 3V, which in turn affected the ADC's ability to read data from the load cell. To remedy this, an approach using two different power sources was tried, where the ADC was powered by a wall outlet at a higher voltage, while the FiPy kept the USB. This resulted in electrical interference throughout the system, because of two different grounds being present in the circuit.

4.1.5 Wiring

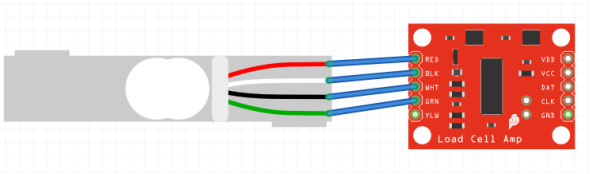


Figure 4.3: The load cell wired to the ADC

4.1.6 Failures

During the first half of the project a faulty load cell was being used, which resulted in major delays of the hardware implementation. During the later half no adequate solution to the problems with the power source could be devised in time for practical testing of the hardware, so it was decided that a virtual load cell would be implemented instead. The function of the

4.2 Software Implementation

4.2.1

Chapter 5

Results

5.1 Discussion

Chapter 6

Conclusions

6.1 Future Work