University Of Tromsø

INF3910

IoT services

Go Green

Authors:
Andreas Nilsen, Tengel Skar
and Alexander Einshøj

date:
May 26, 2017

Contents

1 Abstract

The internet of things is a term which references all embedded devices including their sensors, software and networking capabilities. By utilizing different kinds of networking, cheap micro-controllers with very low power consumption may be used as tools, which delivers real-time data to consumers, and/or function as actuators, allowing them to monitor and regulate processes on their own.

The appliance of IoT services may offload a lot of time consuming work that is performed manually today, improving the cost efficiency of many processes, while also providing more accurate and data than what may be collected manually.

This paper describes Go Green, our greenkeeping project. The goal is to create an end-to-end service which monitors greens on golf courses and displays sensor-data to the consumer, utilizing LoRa/LPWAN technology as a networking tool.

We look into the possibilities of deploying micro-controllers equipped with various sensors and LoRa/LPWAN connectivity on greens in golf courses. We build an implementation of the service, using LoPy - an embedded device, and delve into the details of meshing, implementing some of the core components to better support it, but not the actual meshing solution. This is reserved for a stage after the prototype is deployed and field-tested.

The current version of the service only collects environmental data. In future iterations, it could also be expanded to automate part of the labor, like activating the irrigation system, pumping water into reservoirs, etc.

The environmental factors monitored are general for most types of vegetation, and by implication, the same service should be applicable to various other fields, such as agriculture or even public parks.

2 Introduction

The project was attractive, as the chances of deploying seemed good from the first presentation. Further on, the project provides a functional use which seems interesting to implement from a programming point of view, as well as an economic stance.

2.1 The daily struggle of a greenkeeper

Keeping greens in a pristine condition is a tough job. On a normal golf course, several people are employed solely for this purpose.

The amount of manual labor being done at a golf course is very high. Our solution aims to give the greenkeepers information to help them better decide their daily

priorities. Furthermore, our solution will provide information which is not normally addressed by the work of manual labor, or any other labor for that matter.

2.2 Who will benefit from our solution?

Given that the service provides relevant information, the application of an IoT service on a golf course should benefit both the consumer and provider.

- The product will benefit golfers by golfing on greens in pristine condition, and possibly having information about the state of the greens available to them.
- The employees on the golf course will be able to work more efficiently, by having access to all the information they need in order to keep the greens at a pristine state.
- The owners of the golf club will in turn end up with a more competitive golf course, by being able to optimize the course. It may also improve profits, by minimizing over usage of fertilizers and water.

2.3 The Ideal Application

The ideal state of affairs would be a golf course on which the employees water, fertilize and cut the greens exactly when needed. Embedded devices, with sensors would monitor the environment to provide a stream of real time data to the employees, allowing them to act accordingly. Ideally, embedded devices would monitor other factors, such as pH and salinity in the soil.

2.4 Elaboration

The problem with the current practice is the quality and frequency of information. The work done on the greens is currently based on routine and manual checks. The application of an IoT service could significantly improve the efficiency of said work. Furthermore, constantly checking the PH-value would require a lot of manual labor, which could be significantly reduced.

Cost is an important issue regarding the service. A very high cost for a deploying a complete solution naturally leads to fewer customers, since coverage of a full golf course might be too expensive. Also, in the scope of this project, we have disregarded the use of certain sensors, such as sensors which measure salinity and pH, due to cost limitations. The cost of embedded devices vary largely in price. The LoPy, which is somewhat expensive because of its LoRa module, has a unit cost of around 30 euros. Other embedded devices which feature only a WiFi-module are a fraction of the cost.

Efficiency is also important. In a perfect world, an IoT device will last a full season on a single battery, while requiring very little maintenance during its lifetime.

The LoPy in it's current state has a few limitations related to power saving. A hardware related error has made it impossible to activate deep sleep, although Pycom, the developers, are shipping expansion boards which resolves this issue in the near future.

Currently, a solution using LoPys consumes a lot of power, no matter the configuration. As a result, it is highly unlikely that a field-deployed LoPy will last for long in active state.

This, and the need for flexibility led us to consider the implementation of meshing, by using ESP8266 micro-controllers to reduce the cost, as the price tag is between 1/5 - 1/10 of a LoPy device, while consuming less power. It should also be noted that meshing could lead to lower maintenance and replacement cost of devices, which is far preferable to the alternative of a LoPy solution. Although meshing is not implemented, we did implement some features with meshing in mind.

For practical reasons, the longevity of a deployed embedded device is difficult to estimate. As an example, a device deployed on a green would most likely have to be moved every time the grass is cut, unless the device is provided with a dedicated spot where grass is never cut. Also, golf balls may hit the device, causing potentially irreparable damage.

2.5 Moving on

Following in this paper, an introduction to the technical aspects of the project will be explained. The design will describe how our solution is created, and how we built this end to end service. The implementation will elaborate on the process of developing our project. Following this, our thoughts and alternative options will be explored in the discussion. Lastly, the conclusion provides some of our personal thoughts regarding this project, its outcome, and future plans.

3 Technical Background

The following section provides some information regarding the various concepts mentioned in this paper.

3.1 LoRaWAN / LPWAN

Low-Power Wide-Area-Network (LPWAN) is a type of wireless telecommunication wide area network designed to allow long range communication at low bit rate among things (connected objects), such as embedded devices with sensors, operating on a battery[1]. LoRaWAN is a communication protocol built on said technology, which targets key requirements of Internet of Things such as secure bi-directional communication, mobility and localization services [2].

3.2 Telenor Connexion

Is a service that provides a full interface for registering embedded devices, allowing users to generate LoRaWAN certificates/authentication tokens that compatible devices may use to connect to Telenors LoRa service.

Their web-app provides a view which is highly customizable, allowing display of relevant device data, independent of type of solution. It is clearly intended for developers, but a view for customers could definitely be built solely in the framework, as well. The service provides the ability to forward data received from devices - prior to, or after transforming it - to an external recipient, for example a mail account or a web server.

3.3 Mesh Network

A mesh network is a type of network where a collection of devices, each reffered to as a node, cooperate by relaying data for each other. This allows for a network of nodes with a much greater span than that of a single node. This can be applied both wired and wirelessly. Later in this report we are going to dig deeper into a specific type of mesh network called a *wireless ad hoc network*.

3.3.1 Ad Hoc Wireless Network

An Ad Hoc network does not rely on an existing *infrastructure*[1]. Routers and access points are examples of network nodes that form a static infrastructure. In an ad hoc network, devices collectively build a dynamic network by communicating with surrounding devices.

The advantage of not being dependent on an infrastructure is the possibility of self-configuration. If a node disappears from the mesh, the network may still retain its span, since the network may redirect the data flow through other nodes. Each device can move freely, in contrast to a network with infrastructure. If you physically move a wifi router, there is a chance that connected devices are no longer in range of the router, causing them to lose connection. This of course applies to a device in ad hoc network. But the devices in the network are not dependent on any specific single device, you can be in range of any arbitrary device in the ad hoc network to be connected. This allows devices to create and join network anywhere as long it is in the reach of any devices in the network.

[1] http://www.wisegeek.org/what-is-network-infrastructure.htm

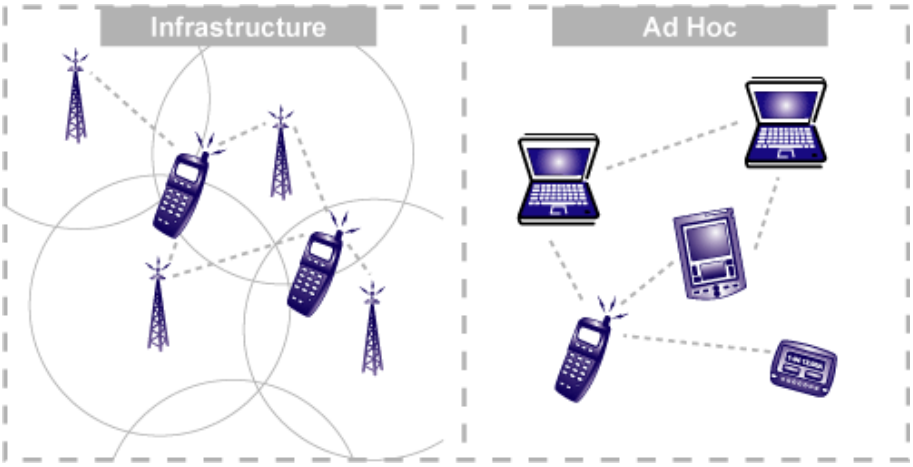

Figure 1: Difference between infrastructure and ad hoc[3].

3.4 Microcontroller

A microcontroller is a self-contained system with *peripherals*[2], memory and a processor that can be used as an embedded system[4]. Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems[5]. This project only uses on the usage of programmable microcontrollers.

- **LoPy** is a microcontroller with wireless communication features such as LoRa, Wifi and BLE[3]. The LoPy is programmable with MicroPython.
- **ESP8266** is a low-cost Wifi microcontroller. It has a full TCP/IP stack[4]. It supports a range of languages. Most notable are C and micropython.

3.5 Sensors

The majority knows that a sensor is a device that responds to some physical environmental input, and creates a output that is readable for humans. Below are some sensor we are going to discuss in this project.

- **SHT10** is a mountable relative humidity and temperature sensor, the sensor can both be applied to air and soil. when used in soil, the sensor need to be protected by a probe. Notice the SHT10 cannot be addressed by the I^2C protocol; however the sensor can be connected to the I2C bus without interference with other devices connected to the bus[6]. The power supply must be in range of 2.4-5.5V

[2]A peripheral device allows for any external devices to be connected to it.
[3]BLE Bluetooth Low Energy
[4]Complete set of networking protocols

- **BME280** is a combined digital humidity, pressure and temperature sensor. The sensor provides both SPI and I^2C interfaces and can be supplied using 1.71-3.6 V for the sensor supply V_{DD}[5] and 1.2-3.6 V[7].
- **ML8511** The ML8511 is a UV sensor, which acquires UV intensity indoors or outdoors. The ML8511 is equipped with an internal amplifier, which converts photo-current to voltage depending on the UV intensity[8].

4 Design

The end-to-end service consists of two main components - the embedded device(s), and the web server with a web application. Telenors IoT service acts as an intermediate component that binds together the communication between the two.

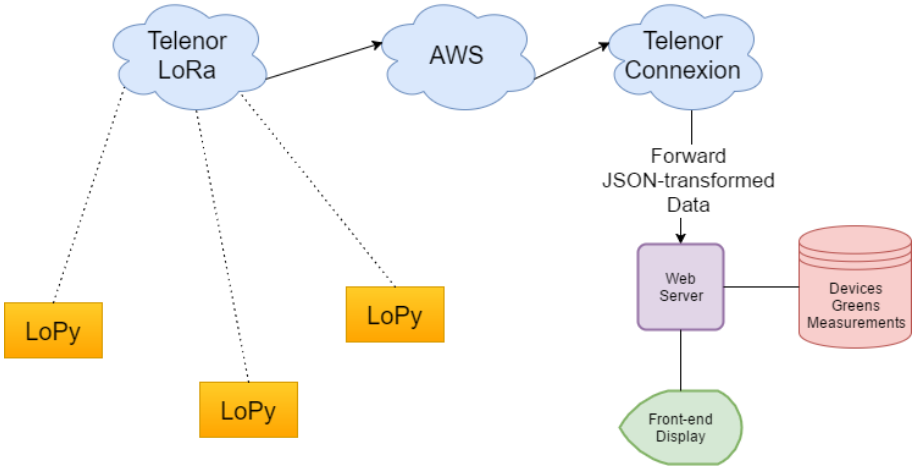

Figure 2: An overview of the end-to-end service

Embedded devices are designed to be deployed in the field, reading real-time sensor data, and transmitting it to Telenor Connexion, where the data is parsed. Here, the data is transformed into JSON-format, and the transformed version is relayed to a custom web server, where it is stored. The webserver's front end provides the consumer - in this case the greenkeeper or other users - with both the latest, and historical environmental data.

Later in this section, the specification of a minimal size protocol built for LoRa data transmission is described. Also, a description of how an extension to said protocol will make it fit for a meshing solution.

[5]Voltage Drain supply

4.1 Embedded devices

This section will mainly cover the details about the LoPy. Since both the ESP and LoPy support micropython, we expect most of the design will translate smoothly to an ESP solution. The principles of GPIO apply to both of the devices, and as a result, the sensors are expected to function equally on an ESP. Given an identical measurement format, the compression protocol will also be identical on the ESP solution.

4.1.1 Software architecture

The main program of the LoPy is designed to allow easy replacement of sensor libraries and LoRa credentials. Also, the lack of connected sensors or libraries will still allow the program to run. This allows the device to support multiple sensor setups without requiring modification of the main program itself.

4.1.2 Boot procedure

When the LoPy boots, two important procedures take place before measurements and data transmission may commence.

Firstly the device attempts to import all sensor libraries, and initializes all sensors, building a list of sensors present in the process. Absent sensor libraries and physical sensors are discarded, allowing the program to proceed as long as at least one sensor is connected.

After sensor initialization, the LoRa credentials are also imported from a file. A LoRa socket and connection is initialized using these credentials.

When connected via a REPL, the device informs the user about missing credentials, libraries, and sensors that are not connected.

4.1.3 Runtime

After initialization, the LoPy goes into a looping procedure of reading measurements from the sensors, packing the data according to protocol specification, sending it, and waiting for a set time interval before starting the next iteration.

When measuring sensor data, the sensor libraries all use a polymorphic method which returns sensor data on a format fit for compression. The measurements from all sensors are added to a list, which is digested by the compression algorithm.

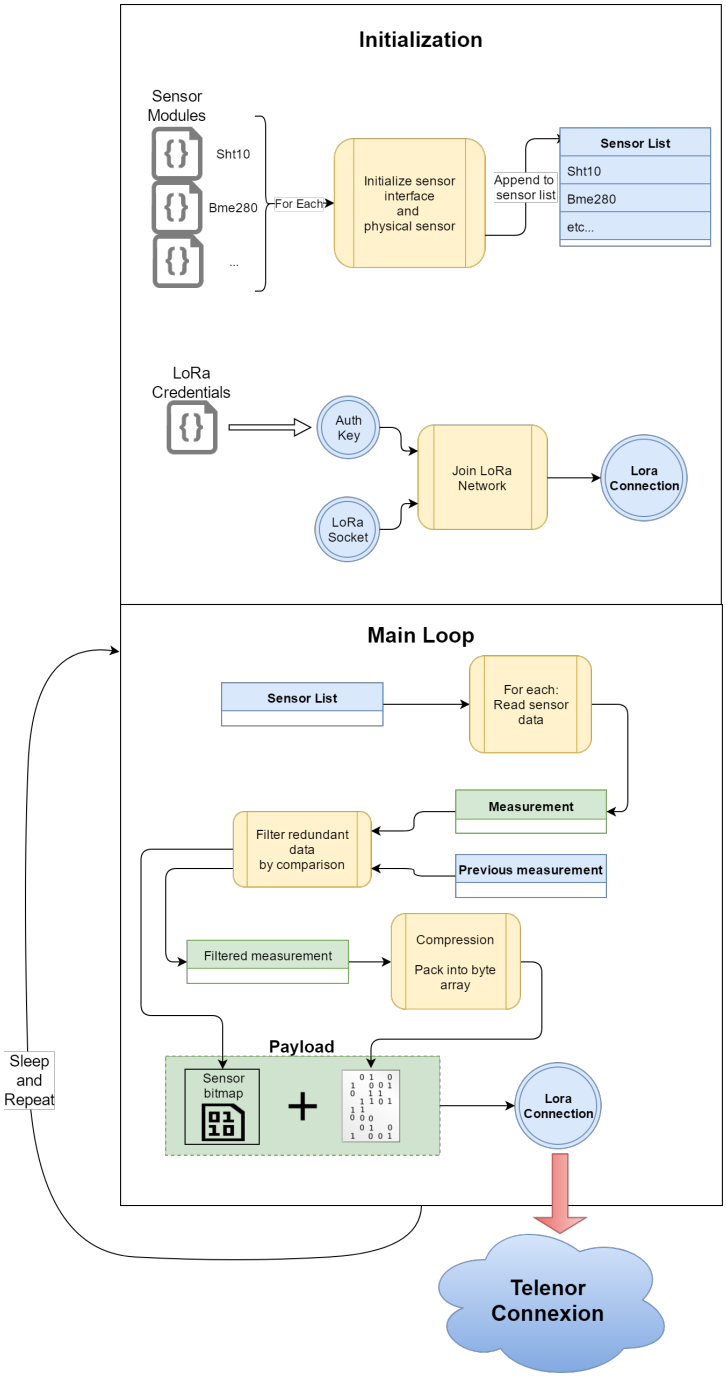

Figure 3: Graphical representation of our solution to the LoPys runtime

4.2 Protocol, data packing and redundancy elimination

The modular design of the device code allows the device to build a bitmap after identifying present sensors. This bitmap is a single byte in size, and a variation of it composes the first byte of the payload.

Whenever a new list of measurements is made, it is compared with the previous list. Measurements that remain unchanged since the previous measurement are filtered from the list, and the corresponding bit in the bitmap is set to 0. If all measurements are redundant, the device skips transmission altogether, and the measurements are discarded. Otherwise, the remaining elements are sent to the compression stage.

In the compression stage, the granularity of sensor measurements is increased to a point that fits the requirement of the application. The use of strings is discarded completely and all values are converted to integer format, yielding a higher information density. As an example, an accuracy of 0.5C for temperature is sufficient for a golf course. Thus, a temperature range between -64 to 63,5C should be sufficient for the purpose of golf courses. This fits perfectly in a 1 byte signed integer. The resulting data is a byte-array containing 1-2 byte integers. This array is appended to the modified bitmap, and transmitted via LoRa to Telenors service.

4.2.1 Scaling the protocol for meshing

To cover the implementation of meshing, ESPs, each associated with a green, would generate compressed payloads as described above. These payloads are sent to LoPys. The LoPys will instead handle the work of tracking which devices it has received data from, generating a bitmap where each bit correlates to a specific green. This bitmap constructs the first few bytes of the final payload, and The ESP payloads are appended to it in sequential order, until the payload size limit is reached. This result is transmitted to Telenors service.

4.3 Telenor Connexion

When a payload is received, the bitmap is parsed, and defines which sensor data is present. The transformation is written in javascript, and does the inverse of the compression that takes place on the device. However, the output is JSON, which is suitable for the server to process.

In turn, the output is relayed via a web hook to the server.

4.4 Server and web-application

How the sensor-data is stored in a database can impact both performance and user experience. The arrows in the figure 4 describes a *belongs to* relationship between the different tables. The idea here is that measures belong to devices and devices belong to both users and grasses. Grasses is a static table containing names on different types of grass on a golf course like teeing ground, fairway, green etc.

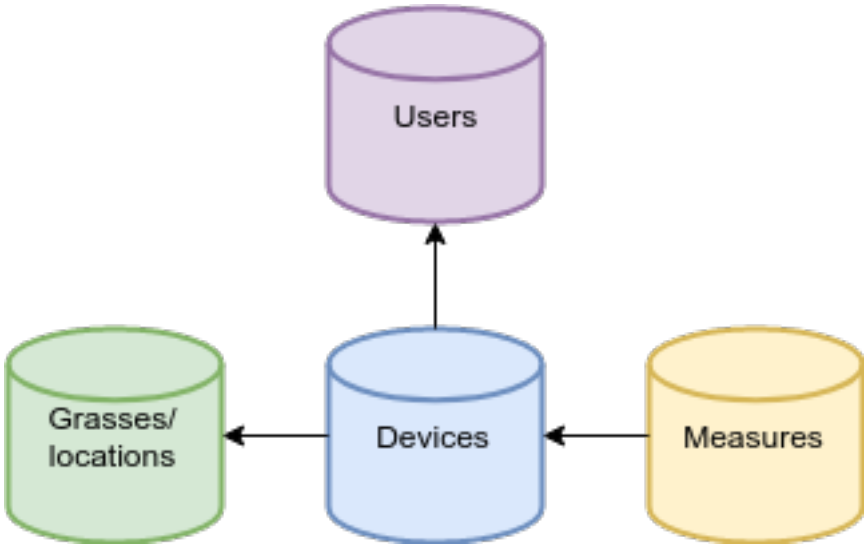

Figure 4: Simple layout of a possible database design.

The concept of the server is to display sensor-data in a orderly and readable fashion for the user. Furthermore, the interface can not exclusively exist of graphs and tables. The user should have the possibility of navigating through the web page, and select measurements from various courses. There exists different ways to display the data, thus the user should have the opportunity to change how data is displayed.

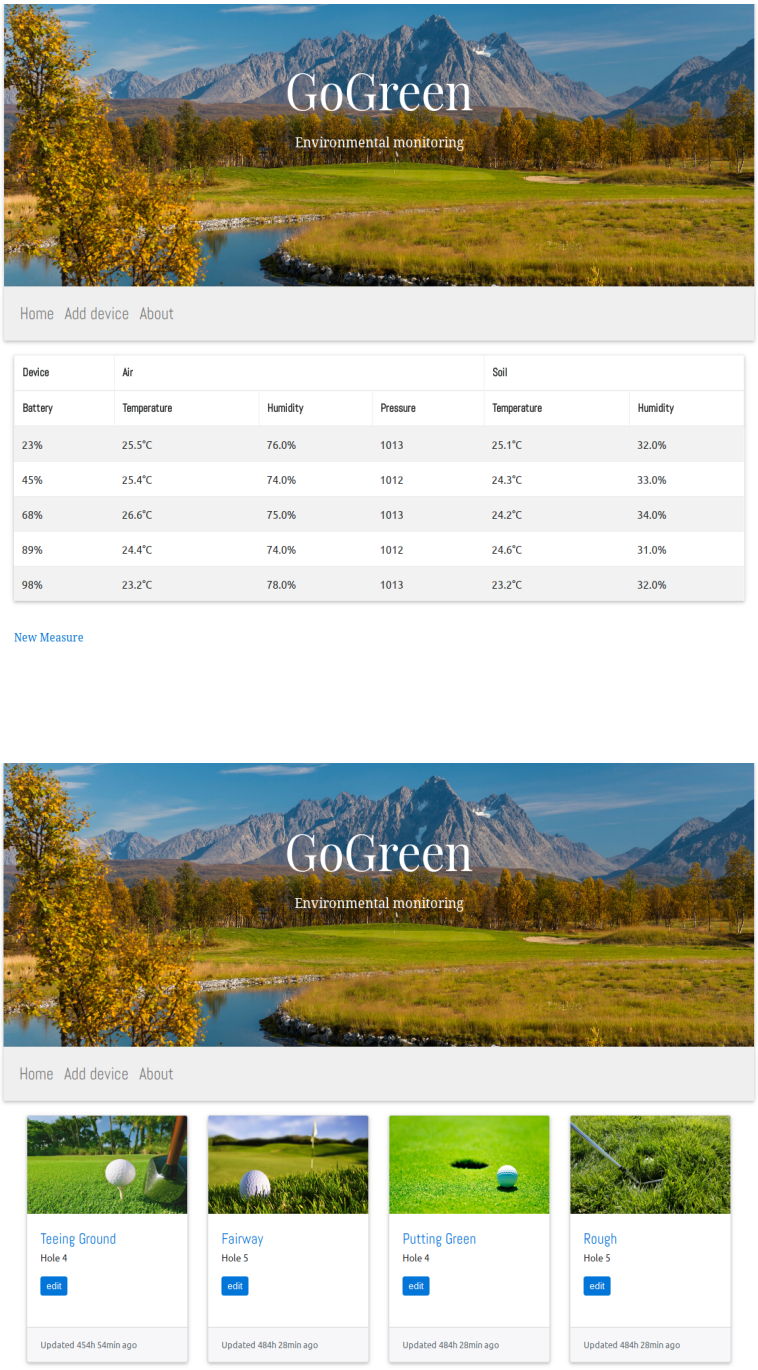

Figure 5: Current design of the web server

5 Implementation

The device solution is implemented with Micropython, utilizing Telenor's LoRa networking and IoT services. The webserver is implemented using Ruby on Rails. Drawing graphs is done in javascript utilizing the *chartjs* library. The server will be deployed on Heroku.

The following sections will cover decisions, discoveries and experiences made in the process of designing and implementing the service.

5.1 Milestone 1: Concept

Milestone 1 revolved around defining our project and service. We found quite an interest in meshing, and wanted a project that could make use of it. We considered the possibilities of environmental monitoring, specifically looking into monitoring air pollution. The decision fell on greenkeeping because we learned that the chances of deploying the service were good.

5.1.1 Determining the sensors required

After choosing our project, we looked into which sensors were needed. Among the types of measurements were air and soil, humidity and temperature, air pressure, and light monitoring. It should be mentioned that light monitoring is not very relevant in the context of Tromsø during summer (when golfing is actually viable here) because the sun never sets.

Other sensors, such as pH and salinity were considered, but looking at the price tag, which starts at around 150 USD for each sensor, left us disregarding said type of monitoring for the scope of this project.

5.1.2 Getting familiar

At this point in time we had no sensors, nor any ESP8266's. We did however play around with the LoPy devices . We were able to generate certificates, and connect our devices to Telenor Connexion, send data, and display the data using Telenors web-app.

5.2 Milestone 2: Design

Early in milestone 2, our focus shifted towards a non-meshing solution. The main reasons were that we hadn't received the ESP8266s we had ordered, and the most sensible action at the time was to build a proof of concept.

5.2.1 Workshops

During the iteration, we attended two workshops arranged by Telenor. At these workshops, we learned how data transformation functions in telenor connexion's service, and found that since it is a javascript solution, it supported binary data transformation. Furthermore, we found libraries that interface with BME280 and SHT10. These libraries were implemented in a manner which would be of use to our project.

5.2.2 Defining the details

We put a lot of focus into the development of a module based software architecture for the devices, with the goal of building a plug and play solution, emphasizing the ease of swapping credentials, and also allowing the device to function with any number of the implemented sensors.

We also began specifying a protocol to minimize the size of payloads. Sending data via LoRa is expensive in terms of power consumption. Given the implementation of meshing, this protocol could be expanded to build single payloads composed of measurements from multiple embedded devices, allowing a single LoPy to function as an endpoint for multiple ESP8266s.

5.3 Milestone 3: Prototype/Alpha stage

During this iteration, the software architecture was composed and implemented. The LoPy prototype was tested, using the air and soil sensors, both locally using a REPL with a computer connected to the device, but also transmitting the measurements using LoRa, verifying that it all came together properly.

5.3.1 Protocol

An early version of the payload compression was also implemented. At this time, the protocol featured no redundancy elimination, and as a result, a device sending six types of measurements per payload yielded a constant payload size of 7 bytes. The results were great considering a non compressed payload with the same data in JSON-format easily could be up to 50 bytes in size.

The LoRaWAN specification does not guarantee the arrival of data packets. Depending on signal reception, the practical limit of a payload's size may range from 50-200 bytes. [9]

Decompression of the data also required some tinkering with Telenor Connexions uplink transform. At this time, the decompressed data output text of no specified format.

5.3.2 End of milestone

At the end of the iteration, the SHT10 and BME280 sensor both functioned properly. We had some issues regarding the UV light sensor(ML8511). It outputs a maximum voltage of 3,3V, which the LoPys GPIO ports are not designed to handle. This would require a voltage divider. Our conclusion was to use the sensor only with ESPs, since its GPIO is built to handle the voltage.

5.4 Milestone 4: beta-version

At this stage, the protocol was further improved. The algorithm now takes the last measurement into consideration, and eliminates measurements that haven't changed. By doing this, everything that is sent to Telenor Connexion may now be considered to be an update of the data, instead of a complete refresh. In practical terms, this evaluates to sending less data, considering certain values won't change much if they're updated often. In an indoor testing environment, using the same six measurement types as before, the average payload size was reduced to around 5,3 bytes. It is worth noting that the transmission frequency was also reduced drastically.

The above-mentioned improvement of the protocol should be useful in a mesh network. Since LoRa's payload size limit is quite low, the ability to combine high-density measurements from multiple devices into a single payload is an attractive feature.

5.4.1 Meshing

As mentioned in technical background, currently, the LoPy is only programmable with MicroPython. After digging into the fork of MicroPython for LoPy, we found a major issue. MicroPython does not support WLAN with ad-hoc mode enabled. This means there is no way to create a wireless ad-hoc mesh network with said device. The ad-hoc mode is implemented in Arduino, so it should be possible to implement using ESP8266. Since no fork of the arduino library exists for the LoPy, we decided to skip meshing, as a standalone implementation would take far too much time.

5.4.2 Meeting with Tromsø Golfclub

At the very beginning of the project, we found and ordered sensors. By the time of this meeting, we had already received the sensors we thought were needed for the project, without speaking with those who were supposed to use the end product, so we were quite anxious regarding our choices. However, it turned out all of our assumptions were quite accurate. They did express a need for PH and salinity sensors, but due to the high cost of these sensors, we didn't make any plans to include these in our project.

The meeting had two goals. The first of those being to what degree the plans we had, fitted with their own vision of what they wanted. The second was to talk about potential practical issues which might arise during deployment, in order for us to

prepare for them.

We talked a lot about the placement of devices and how they should be preserved such that they don't get destroyed by the environment. A specific issue was the difficulty of removing devices located at greens. They have to be easily removable or dedicated to a certain spot, so the golf club can mow the grass without fearing that they're destroying the devices.
With the expressed wish for PH and saline sensors, followed an idea to create an actuator for the water reservoir. However, this would be beyond the scope of our project. The thought will carry on should the project be further developed at a later stage.

6 Discussion

During the course of this project, a variety of issues and ideas were discussed. The following section will elaborate on our thoughts of them.

6.1 A financial analysis

It should be mentioned that embedded devices with LoRa/LPWAN modules such as the LoPy are much more expensive than simpler embedded devices featuring only WiFi and Bluetooth. By comparison, a LoPy has a unit cost of around 30 euro, while an ESP8266 has a unit cost of 4-7 euro.

6.1.1 Without meshing

Our chosen BME280 has a unit cost of around 7 euros. Without an implementation which supports meshing, one unit per LoPy is necessary. The SHT10 we used, had a cost of approximately 15 euros, and the ML8511 has a price tag of 3 euros.

A salinity sensor costs around 100 euros per unit. Given how expensive it is, it's of high importance that it makes the best possible use of itself, should it be deployed on or near a golf course. It should probably be deployed at the water reservoir or its intake, as it may prove most useful in said locations.

A PH sensor for measuring the PH value of the water used on the course, would cost about 130 euros. Only one of these sensors should be needed, measuring the intake of water used around the golf course.

The minimal cost of a full deployment, excluding the salinity and PH sensor, would cost approx. 62 euros per green covered. Of that, 30 euros per green is the LoPy, while the remaining 32 euros covers the cost of ESP8266 and all the sensors mentioned.

To cover an entire golf course, 18 greens, the total cost would accumulate to more than 1100 euros. The minimum cost of embedded devices excluding all peripherals, totals around 540 euros. Should one choose to include both PH sensor and a salinity sensor, the total would be around 1350 euros minimum.

The numbers above shows the financial impact of opting for PH and salinity sensor, as it would increase the total cost of devices by more than 20%. This is a large enough number for it to be a significant decision to make, whether to include one or both of the sensors.

6.1.2 With meshing

The implementation of meshing adds a lot more flexibility in terms of deployment and the location of sensors. Without meshing, there really aren't that many options regarding where to put the embedded devices, as one must be located at every green anyways.

Meshing provides the ability to cover multiple greens by deploying multiple ESP8266, and a single LoPy. In a perfect world, the solution would only require one LoPy for the entire course. Since a golf course may span a large area, it may be unfeasible to cover the whole course with a single mesh network, and the amount of data transmitted by such a net may exceed the maximum LoRa data quota of a single LoPy. A more realistic solution could cover the whole course by splitting it into 4-5 subnets, each having a separate LoPy device. Assuming 5 ESP8266's are needed to cover 3 greens, 30 would be needed in total.

The LoPys accumulate to a cost of 150 euros. Sensors are still needed on all the greens, so the cost is a static 450 euros regardless of the routing protocol. The cost of five LoPys, 30 ESP8266, and the sensors needed for 18 greens, total at around 800 euros.

The reduction in cost is quite significant, compared to not using meshing, and is an important reason as to why we chose to focus on the possibility of meshing. The deployment cost is significantly lowered if meshing is implemented.

However, the cost of maintenance is uncertain and may be larger, considering the increased number of total devices. The good thing is, ESP8266's are cheap to replace, compared to LoPy's. However, the manual labor needed to replace these devices is certainly also a cost to consider. At this point it impossible to estimate without having both solutions tested in the field. From a pure economic perspective, without considering the cost of labor to replace the devices, four ESP8266's can be replaced at a lower cost than a single LoPy. That is a notable difference.

6.2 Bitmap protocol

The bitmap protocol eliminates redundancy by not re-sending data if it didn't change since last time frame. this allows for more frequent transmissions of data, and in turn, more reliable data. In the case where no measurements has changed, the device saves

power as it does not need to send the redundant data. This protocol allows for a broad spectrum of use cases, way beyond the scope of this project as well. When handling a network protocol in which one of the biggest limitations is the payload size, this bitmap protocol serves as an extremely efficient solution.

Computation on-chip is cheap compared to sending data over LoRa. sending the same data in JSON format data would be in the range of 20-50 bytes per payload, our solution is a maximum of 7 bytes. This should theoretically reduce power consumption, since computation on-chip is magnitudes cheaper than transmitting data across network.

6.2.1 Future plans

A LoPy-based solution will hopefully be deployed in early June, given that Tromsø golf club isn't covered in snow. This will not be a full deployment, but rather on a test-basis to achieve proof of concept in practice. Depending on the results of the that deployment, further research into meshing might take place during the summer of 2017.

6.3 Choosing a solution

Following is a discussion on the pros and cons of both a LoPy based solution and a solution based on meshing.

6.3.1 LoPy based solution

A LoPy based solution is good because of the simplicity it provides. The total amount of devices to maintain, compared to a solution with meshing, is lower. The biggest drawback with connecting all the sensors to a single LoPy without the extended use of ESP8266's, is that you want to measure multiple environmental factors. This restricts where we can place the LoPy as discussed earlier. Another attractive feature provided through this solution, is how easily implemented it is compared to meshing, which will now be discussed.

6.3.2 Meshing

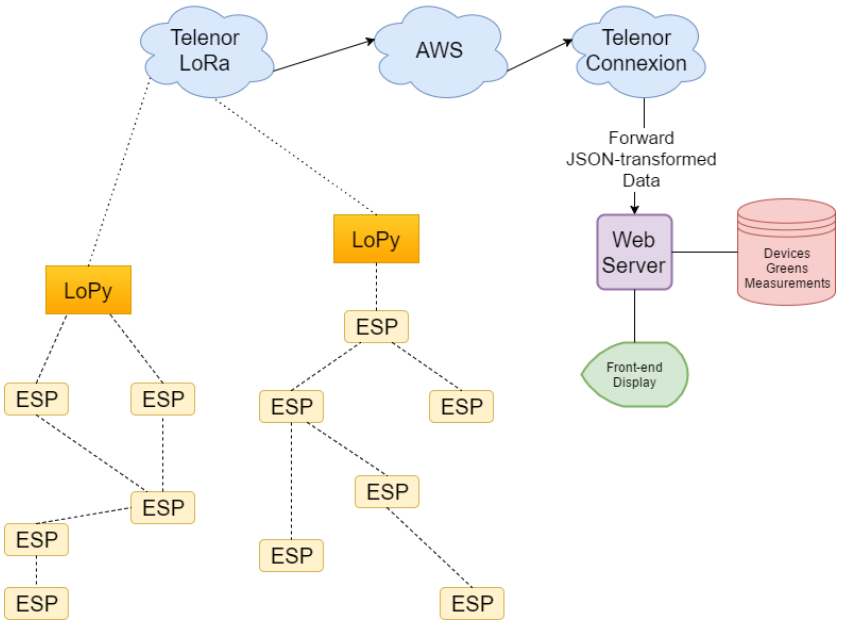

Figure 6: Example of an end-to-end service architecture with meshing.

In the early stages of the project it proved difficult to understand how much we could benefit from meshing devices on a golf course, because of the long distances. However, during the conversation with the golf club, they saw some severe issues with a solution where all the sensors is connected directly to the LoPy. The green-keepers aerate[6] the green biweekly. The LoPy and the sensors will be destroyed if they reside in the soil during this process. The golf club said the devices can be placed under ground in containers, but then the ML8511(UV-sensor) sensor would be rendered useless, and the BME280 is rendered useless underground.

Assuming meshing is implemented, we connect the SHT10 sensor on a ESP8266 and place it in a container underground, before connecting both a ML8511 and a BME280 to another ESP8266 and place that above the ground. Assume we do this on two neighboring greens. If we strategically place one LoPy in the range of these two greens, the LoPy will mesh with the other devices on the greens and send sensor-data that correspond to two greens.

If we can create this scenario, not only will the golf club will save money, but every time they aerate the green they do not need to physically move the LoPy. This solution can also apply to a single green. By only connecting suited sensors on the ESP826, we are granted more flexibility.

[6]https://en.wikipedia.org/wiki/Aeration

7 Conclusion

We have built an end-to-end service using LoRa/LPWAN for networking. The solution uses LoPy devices as data collectors, where compressed data is sent to Telenor Connexion before being decompressed. From there on, the data is made human readable into JSON format, which can easily be displayed on a web page.

Regarding our end result compared to our original idea, we have been mostly successful. Despite this, some limitations made us realize that meshing would be too time consuming to implement in the scope of this project.

Through the experience gained from working with this project, we've found IoT to be an attractive field, with a plethora of applications. we hope this paper has provided an insight on the subject of designing an IoT solution, its current limitations, but also the great potential it has. We may certainly conclude that it is a great tool, which may only grow in usefulness once more fields start applying it.

References

[1] (). Lpwan, [Online]. Available: `https://en.wikipedia.org/wiki/LPWAN` (visited on 05/25/2017).

[2] (). Lora alliance™ technology, [Online]. Available: `https://www.lora-alliance.org/What-Is-LoRa/Technology` (visited on 05/25/2017).

[3] (), [Online]. Available: `https://cdn.sparkfun.com/assets/learn_tutorials/5/3/4/network-infrastructure-vs-a.png`.

[4] (). What is a microcontroller?, [Online]. Available: `http://www.futureelectronics.com/en/Microcontrollers/microcontrollers.aspx` (visited on 05/25/2017).

[5] (). Microcontroller?, [Online]. Available: `https://en.wikipedia.org/wiki/Microcontroller` (visited on 05/25/2017).

[6] Sensirion, "Datasheet sht1x", pp. 1, 5, 2011.

[7] B. Sensortec, "Bme280 combined humidity and pressure sensor", p. 3, 2015.

[8] L. S. Co., "Ml8511 uv sensor with voltage output", 2013.

[9] (). Limitations: Data rate, packet size, 30 seconds uplink and 10 messages downlink per day fair access policy., [Online]. Available: `https://www.thethingsnetwork.org/forum/t/limitations-data-rate-packet-size-30-seconds-uplink-and-10-messages-downlink-per-day-fair-access-policy/1300` (visited on 05/25/2017).