Using sensor networks and cloud technology to capture and visualize soil moisture data.

A technology feasibility study

Erlend Westbye

Master Thesis, Programming and Networks
60 credits
Department of Informatics
Networks and Distributed Systems

November 14, 2019

Abstract

This thesis describes building a small scale wireless sensor network implemented using commodity hardware and open source software. The sensor network captures soil moisture and temperature data, and distributes the data to a web application programming interface running in a cloud environment. The goal of the implementation is to answer whether this system is a feasible way to reduce excess irrigation in agriculture through improved information and visualization. The system collects, stores and visualizes data and can support high throughput of data. The thesis describes both the system design and the implementation in detail. As well as describing how the modern technology paradigm cloud-computing can aid in the development and deployment of the system described in this thesis. The evaluation describes how the implemented system is a feasible way to address excess irrigation, and describes water management in agriculture as an information and management problem above all else. Through implementing a proof of concept system for monitoring soil moisture, the thesis will demonstrate how technology can be used as an integral part of monitoring and reducing excess irrigation.

Figure 1: Hjemly farm Osen, Hedmark

Contents

Chapter 1

Introduction

In this chapter follows a review of motivation both from a technical, environmental and personal perspective. It will also set the scope for the thesis and set requirements for the system described in the thesis. It will also include a brief summary of central findings for the thesis.

1.1 Why is high water usage a problem

Many are like me and turn on the shower without giving much thought to the amazing innovation that has gone in to getting the water running through the pipes in to my apartment. But I got to admit, I think less about the water it self. Especially in a country like Norway where water is abundant, and there is no significant visible cost attached to turning on the shower as both water and electricity are cheap here.

Although we rarely think about it, water is a finite resource. Non-frozen Fresh water makes up 0.76 % of the world's water supply [40, p. 13]. This number really sets the amount of available water in perspective. I will not be arguing whether the commodification of water is the worst thing that can happen to modern society, whether the free market knows what is right for the world, or how much of a toll agriculture is taking on the atmosphere of the earth. This thesis will look at the feasibility of using modern technology to reduce water usage through improved information availability in one of the most water intensive industries; Agriculture [17].

Water is high on the agenda. The United Nations has a list of sustainable development goals, where water is one of the items on the list [37]. The OECD has also put forth water as one of the important things to put on the agenda. In a 2018 note they also mentioned the gathering of information as one of the important steps in improving the situation.

"Improve information systems on surface and groundwater quality and flows, help to assess risks, and implement programs tailored to specific challenges" [27].

Norges Bank Investment Management (managers of Statens Pensjonsfond Utland) has also made sustainable water management one of their goals for sustainable investing.

"Companies should recognise the business' water impact, commit to sustainable water management and, as relevant, have a clear water management strategy" [24].

1.2 Drought in Norway 2018

During the summer of 2018 Norway experienced severe drought. The lack of moisture in the soil resulted in severely degraded yield in livestock feeds grass and grain. This lead to severe financial losses for farmers in south and middle Norway [26]. So even water rich Norway has seen that we are not immune to the changing climate and the effect that water shortages can have on our domestic supply of agricultural goods.

1.3 Large scale data streaming

The biggest cloud vendor in the world, Amazon Web Services, was started in 2006 [13]. In the last 13 years the rate of innovation in this space has been staggering [3]. There are many vendors that have achieved impressive data streaming capabilities (Netflix, Bloomberg, Facebook). There are many services for gathering, processing and storing large data sets in different industries.

Examples of a service that has massive data stream systems are found within the finance industry. Where companies like OneTick can offer solutions that can stream and persist every single trade that takes place on an exchange [29]. This gives a good idea of the big data capabilities that are implemented in cloud. Cloud vendors provide services such as hosted data bases, data streaming platforms and data lakes. Developers are getting used to having immense computing capabilities at their finger tips, and data streaming is one of the things that could be implemented in cloud.

1.4 Technology adoption in farming

The farming tradition in Norway is rich, and driving through the Norwegian country side shows the large focus we have had on farming as a nation. Farming is an area that contains some incredible innovation on many scientific fronts, we have modified the genes of plants to make them more drought resistant and resistant to pesticides [10]. It can therefore be argued that agriculture is an industry where technological innovation is ingrained in the activities that farmers take on every day. We also see that farming equipment is going smart. We have seen international media coverage on farmers going so far as hacking their farming equipment [23]. Agriculture is an industry that has needed to adopt new technology constantly, and it can therefore be argued that this is an industry that is eager to take on new technology and primed for digitization and innovation.

1.5 Using sensor networks and cloud technology to manage water usage in agriculture.

In the context of this thesis I am interested in digitalization through monitoring of agricultural variables. In this thesis I have decided to focus on soil moisture as a proxy for water consumption. The research paper presented by Amy Lilienfeld and Mette Asmild [20] gives a clear overview of a method for quantifying excess irrigation. One of their central findings is that water consumption and type of irrigation system has no clear relationship. This points to a management and information problem, rather than a pure mechanical one. Other interesting findings that further supports the fact that excess irrigation could be a management and information problem is the fact that the age of the farmer had more to

say for the water consumption of a farm than the irrigation system. These two findings can suggest that through simplified information delivery and digitization of agriculture can have an impact on excess irrigation. Mismanagement of water can lead problems such as over pumping of aquifers [20], and water management in agriculture is an area that improved use of information technology, and reduced cost of information gathering, can be used for good. With this in mind this thesis will try to address excess irrigation through providing farmers with an intuitive overview of the current state of their soil.

The scope of this thesis is to answer the three questions listed below, in an effort to address excess irrigation through improved information availability and visualization.

- Is sensor technology a reasonable way to address excessive irrigation?
- Can a system for monitoring soil moisture be implemented with limited resource use?
- Is it practically possible to implement a system for monitoring soil moisture?

Keeping this in mind this thesis will address and answer all of these questions. From an intuitive standpoint one could think that the answer to all these questions are yes. However, this thesis will show that these questions are not just true from a theoretical point of view, but also something that can be implemented by a small team with limited resources.

1.6 Approach

To gain insight in to what factors could be monitored and how to architect a system that could support farmers in making good water management decisions a proof of concept was built. In addition to building a proof of concept, I went to meet with a cattle feed farmer to discuss monitoring and what factors are of interest to them. I also had a meeting with a leading Norwegian manufacturing automation firm to learn about their sensor offerings as well as how they develop their monitoring platform. In addition to these activities, I learned about the topic through reading previous work on the subject.

With this mixed approach of learning by doing and reading about the subject at hand, the thesis is to discuss the challenges and problems that have to be solved to improve water management in agriculture by improving information availability. And how IoT and cloud technologies can help in improving the water usage in this sector through building a data gathering and visualisation system.

1.7 Personal motivation

There are revolutions within technology that has enabled this thesis. The first is the cost and prevalence of micro-controllers. A micro-controller is a small general purpose computer that typically consume little power and can easily be used in an embedded context. The two major micro-controllers, the Raspberry-PI and the Arduino, both ship with "general purpose inputx output pins" (GPIO) meaning that these boards can both process data from the real world and the digital world, provided that you feed it with data using sensors. This link between the real world and the digital world is something I find deeply facinating and it is what truly led me down the path of becomming passionate about programming. Another revolution has come in the availability of on demand computing resources through what has been dubbed "the cloud". When I started programming in 2013, cloud was already established as a good way of hosting applications, and is the deployment model I have used for most of my career. Cloud is a true enabler of proof of concept implementations, as the upfront cost of implementing advanced system is reduced drastically as you would typically only pay for the resources you actually consume.

This fascination with cloud and combining the real and the digital world is something that has given me a lot to think about in the last five years. And the reason why I wanted to build this system is to investigate how I can leverage technology to improve and help managing processes around irrigation in agriculture. I had long had an idea of starting a smart farm at some point, and I think that this backdrop has given me ideas for how to implement this in a small scale outdoor operations at my farm in Hedmark where my wife and I will grow grass for cattle feed. Although I did not own this property at the beginning of this thesis it has given inspiration and motivation through 2019 and I will go in to detail on how I envision deploying this system there in the subsection marked "Deployment conceptual".

1.8 Quantifying the requirements of the system

The requirements for this system to meet the purpose of the thesis, and being able to answer my central questions are based on the challenges described in the research paper presented by Amy Lilienfeld and Mette Asmild [20]. In addition to technical requirements that became clear in the design of the proof of concept, and the planning of this thesis. The fact that the research paper mentioned above establishes a baseline by looking at the best actors, using them as a benchmark is a natural choice when the cost of gathering data is high. This thesis will show that the cost of gathering moisture and general plant related data can no longer be described as expensive.

The system will need to handle concurrent requests from a wide range of sensors at the same time. This is important for a production context as the data gathering APIs should not be a limiting factor in the system. To achieve this the system should be able to scale horizontally and be primed for hosting in the cloud.

Documenting a low cost, high throughput system is an integral part of answering whether this system can be incorporated in the irrigation process and help farmers make better choices for water usage. The system will need to visualize the data to a standard where it is intuitive for a farmer to see the water needs of a plant.

To summarize, the system will need to be able to gather data at a low cost, handle a high number of sensors at the same time and visualize the data in an efficient manner. If all these are achieved, this system could be an integral part in the management of irrigation for any farmer that uses it, and contribute to reducing water usage in agriculture through enabling better decisions through improved information availability.

1.9 Research method

This thesis aims to conduct a feasibility study considering the cost of an implementation in cloud and on premise. To research the feasibility, a monitoring system has been designed and implemented with cheap and commonly available

hardware. There has been written custom software to get a grip on the amount of data this system would produce on a larger scale. Taking in to consideration what data is relevant, and how to gather data from multiple sources. The thesis will look at the costs and technology considerations of this system. Other important factors is the importance of data accuracy and data integrity. A large portion of the system that is discussed in this thesis has been implemented to learn about the networking and hardware challenges that comes along with a distributed wireless sensor network. As a part of this implementation, different cloud platforms have been tested to provide insight in to the pricing and resource consumption in the cloud.

Setting up the sensor networks is a challenging task as the developer is doing the work of a programmer and a product developer at the same time. The thesis aims to show that it is feasible to build this type of system yourself and that there is room for an open source alternative to the commercial products that are discussed in this thesis. I sought out to learn about the subject by building a functioning prototype. It is one thing to read about prior experiments, but a whole other to build it out yourself.

Information about similar systems has been gathered, as well as information on the state of the art open source and commercial products in this space. It is also important to mention the cloud aspect of this proof of concept as it is one of the key cost drivers. The use of the vendors Microsoft, Amazon and DigitalOcean has been tested.

1.10 Central findings

1.10.1 Software

Through exploring the feasibility of a large scale agricultural sensor networks I have discovered that this type of software can be implemented by developers with modest professional experience using open source frameworks and public information. The implementation overhead can be kept low as the software built is based on open source implementation of different systems. As is seen often with a systems development, the code-foundation is out there in the public via open source, and it is possible to combine in to new products in a relatively

short time frame. However it is important to note that implementing software is never trivial, and it is important to acknowledge the community resources and open source software that helps developers along the way. One central finding is that the willingness to share information and help other developers is a major part of why it is possible to implement this type of system.

1.10.2 Hardware

The price of the hardware was the biggest discovery made in the hardware portion of this implementation. The ubiquity and availability of fit for purpose and general micro-controllers has been a huge help in the implementation of this proof of concept. The usage of the RaspberryPI and the NodeMCU micro-controllers are especially simple as there are active communities for both. As an example I got electrical engineering help via the internet on a forum by posting a description of a problem I was facing. This example shows the power that one individual is given in the hardware space through online collaboration and open source platforms. Through the thesis work it was also found that the system could be implemented using commodity hardware bought from cheap sources, and the availability and level of service offered by platforms such as Ebay and Alibaba are key enablers when doing prototyping development where physical hardware is required. But when it comes to managing battery power and implementing a system using stripped down versions of the esp8266 [12] the challenges are no longer trivial to a developer with basic knowledge of electrical engineering and physical hardware manufacturing.

1.10.3 Cloud

Three different cloud vendors where tested and found to be more than adequate for the purposes of this proof of concept project. The proof of concept implementation used a common Linux distribution (Ubuntu), MongoDB installed on said server on the platform DigitalOcean. In later times hosted services like DynamoDB from AWS and documentdb from Azure where also tested. The hosted services from Microsoft and amazon had a larger than expected impact on the cost of implementing this type of system.

1.10.4 Cost

A basic implementation was deployed for a low cost, and has been kept the deployment running at approximately $ 5 per month. The hardware is also relatively cheap, depending on the type of sensor used. With the most basic sensor described in this thesis the cost was approximately $ 2.40 per sensor package (mote), and with the more advanced sensor described in the implementation part of the thesis the price was around $ 15. Ordering bulk quantities of the same hardware would yield a lower price.

1.10.5 Feasibility

From a technology standpoint, a system for monitoring and visualizing soil moisture is possible to build and can be implemented without excessive technical competence, and without high development and hardware costs. From a water use perspective the answer is more complicated but based on the theory that water usage is tightly coupled to management routines, it can be argued that this would be of great help to farmers that want to monitor and improve their water usage through increased information and visualization.

Chapter 2

Background

This chapter contains information that has been deemed as relevant knowledge for reading the following chapters. It will cover background on excess irrigation benchmark and analysis. In addition to technical information that is relevant for understanding the implementation of the system.

2.1 Is over watering a management or a mechanical problem

The application of more water than a crop can use, or over-watering, is usually the result of lack of knowledge about soil water content or crop water demand [20]. This statement highlights the fact that knowing the moisture content of the soil could be integral in reducing the over-irrigation of crops. Quantification of the extent of over-watering has been shown to be valuable [20], this further supports the claim that quantifying the soil moisture content would be a valuable tool in reducing over-watering in agriculture.

It is relevant to provide some context on the common types of irrigation systems that are deployed. These will wary across geographical locations, but it is from a intuitive standpoint fair to assume that irrigation is being done in a similar fashion across the world. Included below are the systems discussed in the paper by Amy Lilien-feld and Mette Asmild [20] as well as describing some of the other common irrigation methods that can be relevant to know about. The two main differences are: Using rainwater or actual irrigation.

Rainwater irrigation is an important part of farming and following the weather has a high importance in farming. As an example drought can decimate the yield of a crop. We saw this happen in 2018 with livestock feeds like grass here in Norway. In these situation water management becomes especially challenging as wells and other natural sources of water can get depleted. Rain-fed farming is the natural application of water to the soil through direct rainfall. Relying on rainfall is less likely to result in contamination of food products but is open to water shortages when rainfall is reduced [9].

Irrigation systems include but are not limited to surface irrigation, sprinklers, drip irrigation and center pivot irrigation. The main differences for the types of irrigation is the precision of the irrigation and the cost and the labour involved in doing the irrigation activities. Intuitively one could think that there is a high correlation between the type of irrigation used and the amounts of water used. As an example you could think that any irrigation that involves flooding would result in excessive irrigation. But as argued by by Amy Lilien-feld and Mette Asmild [20], a major finding of the study is that there is only a weak relationship between irrigation system type and the level of excess irrigation water used [20].

This is interesting to the research case as the implication would be that the resource intensive and costly exercise of changing or replacing the irrigation system used would not automatically result in a reduction in water usage. This highlights that gaining insight in the actual water amount requirements over time could be valuable input to the management process of a farm. One of the challenges highlighted by the paper [20] is that the gathering of agricultural data is costly and inaccessible to a large portion of farmers. Given that this paper is already 10 years old, the working implementation of the soil moisture monitoring system shows that the price of gathering agricultural data no longer has a high cost relative to the price of doing mechanical upgrades to an irrigation system. It is argued that irrigation systems are not clearly inefficient and the management of irrigation may play a large role in the efficiency of the water use [20].

The conclusion that the management plays a significant role in excessive water usage is especially interesting given that one of the primary goals of this thesis is to conclude whether or not a system for collecting and visualising water con-

sumption using WSN and cloud is feasible. This conclusion by Amy Lilien-feld and Mette Asmild [20] is a big motivating factor and gives a good foundation to discuss the feasibility and value of a system for visualizing the soil moisture, and exposing the moisture data in a simple way.

2.2 Cloud Computing

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[22, p. 2] This is the National Institute of Standards and Technology's (NIST) definition of Cloud Computing. NIST further define the concept with five characteristics, three service models, and four deployment models [22]. The five characteristics are:

- **On-demand self-service** A customer can provision computing capabilities at any time without any human interaction with the vendor.
- **Broad network access.** Resources should be available for access from a wide range of devices.
- **Resource pooling** The vendors resources should be pooled and served to customers based on demand.
- **Rapid elasticity** Computing capabilities should be elastically provisioned to provide rapid scaling.
- **Measured service** Resource usage should be monitored, controlled and reported providing transparency for both the vendor and the customer.

There are a multitude of different service models, but some of the prominent ones are the once defined by NIST:

- **Infrastructure as a Service(IaaS)** "Provision of processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications."[22, p. 3]

- **Platform as a Service(PaaS)** A platform for application deployment giving the user the ability to deploy applications created using programming languages, libraries, services, and tools supported by the provider.
- **Software as a Service(SaaS)** A software provided to the customer running on the vendors infrastructure. The application should be available from various devices.

One of the key technologies that has made the modern cloud possible is virtualization [41]. Computer virtualization refers to practice of running operating systems on virtual hardware. This is achieved by splitting the resources of physical hardware between different instances of operating systems. This is done by installing software known as a hyper-visor on the hardware instead of an operating system like Linux on a computer. The hyper-visor allows for operating systems to be installed on top of the hyper-visor, so that one machine or pool of resources can run many operating systems at the same time [36].

2.3 Internet of Things

IoT is the concept of connecting “things” to the Internet to gather data and perform actions. Examples of uses for IoT range from manufacturing to agriculture. And the concept is often looked at as a way to connect everything from fridges to juice presses to the Internet. The recent development that has made IoT more available is the low price and number of of WiFi, Bluetooth and radio enabled micro-controllers. This means that any sensors with a compatible interface can be connected to the Internet at a low cost.

The high volume and low time to market of many IoT products has presented a range of challenges like privacy and network security. There has already been large scale issues with connected appliances. The hacking of the connected light bulb Phillips "Hue" [25] is one example of a popular IOT product being hacked. Despite these concerns the number of IoT devices is rapidly increasing [11].

2.4 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are ”a network of battery-powered sensors interconnected through wireless medium and are typically deployed to serve a specific application purpose” [28, p. 68]. Technological advances have made small, low cost, low power and highly customizable devices commercially available and has improved the viability of large scale sensing networks with a large number of intelligent sensor nodes. A sensor node, nicknamed “mote”, in a WSN typically equipped with one or more sensors, a sensor interface, processing units, transceiver unit and power supply [14].

For many years the trend in WSN topology has been a network comprised of several motes and one or several gateway nodes. In this type of network the motes communicate with each other and the gateway node using a radio frequency communication technology like ZigBee or Bluetooth. The gateway node can have several functions including computation and Internet connectivity. The motes in this type of network can not communicate with the Internet directly and relies on the gateway nodes for this, making the network a closed or proprietary system with limited communication to the external world.

Current trends are pushing WSNs in the direction of the Internet of Things (IoT) using the Internet Protocol (IP) to connect every mote to the Internet [21]. Connecting individual motes to the Internet using WiFi enables the utilization of IoT platforms and other resources in Cloud services. In some cases applying a gateway based architecture to an IP-enabled and WiFi interconnected WSN can be the natural choice, especially if the gateway node(s) are able to perform challenging computing tasks.

Fog Computing brings the resources near the underlying network [1]. Compared to the Cloud paradigm’s centralized computing, storage and networking resources. Fog Computing pushes these resources closer to the edge of the network, closer to the user. For a system that demand both low latency and extensive computation, a Fog architecture would be a natural choice. In context of WSNs this means utilizing a Smart Gateway [1].

2.5 The role of IoT and Cloud in modern information technology

The Future Internet is pushing the world towards a state of ubiquitous computing, connectivity and sensing. The concept of Internet of Things(IoT) plays a big part in this paradigm shift. IoT explains the shift in the hierarchy of the Internet where the number of "thing" entities outnumber the number of human entities and makes humans the minority of generators and receivers of traffic. IoT also calls for an increase in Machine to Machine communication (M2M). One of the most important technologies of the IoT is Wireless Sensor Networks (WSN). A WSN is ”a network of battery-powered sensors interconnected through wireless medium and is typically deployed to serve a specific application purpose”[28]. To fully benefit from the ubiquitous connectivity of the IoT computation power, storage and networking infrastructure must enter the equation. Cloud Computing offers these capabilities and the utilization of IoT in conjunction with Cloud Computing is the direction the Future Internet is pushing.

This combination of technologies could be applied in an infinite number of scenarios for better monitoring, data collection, analysis, forecasting and decision making. Developments in technology and in the prices of Cloud services means that these technologies are now available to almost any business that can make use of them. Agriculture is an area where the potential is tremendous for these technologies to work together, especially from an economic and environmental stand point. Agriculture is an area which highly relies on the climate in which it operates and must therefor adjust to the shifting nature of that climate. Knowledge of this shifting nature is key for successful agriculture. Environmental monitoring, data collection, analytic, forecasting, automated systems for irrigation or fertilization, plant health monitoring and frost detection are all scenarios that apply for both conventional and urban agriculture where these technologies can supply crucial information for important decision making.

2.6 Integrating WSNs with Cloud services

There are two main approaches to the integration of WSN data with a Cloud service. Which one suits the network depends on the architecture of the WSN.

The cloud providers AWS and Microsoft offer an IoT platform. Amazon's "AWS IoT" and Microsoft's "Azure IoT Suite" both offers connectivity for most devices, graphic remote resource monitoring, data storage and analysis, data stream analysis, security and application platforms. These platforms offers a broad array of elastic resources and can be customized to accommodate a WSN. There are also more specialized services such as Thingspeak, Xively, Carriots and Kaa aimed only at IoT and might not offer the same underlying infrastructure as the big Cloud vendors.

It is also possible to create a more ad-hoc solution using the resources of the Cloud. Using IaaS services from a Cloud provider it is possible to create a service with the specific resources the network needs. K. Lee *et al*. [19] explores a Cloud service for WSNs built on Amazon EC2 virtual machines. The article demonstrates "how sensor networks can be combined with Cloud Computing to allow the offloading of resource-intensive tasks to the Cloud"[19, p. 7].

2.7 Application layer protocols

One of the key issues in IoT and WSNs is power efficiency in devices, given that a large scale deployment would be severely compromised if the batteries of the nodes in the network had to be replaced often. Application layer protocols has an impact on power consumption as communication is the most power hungry task the devices perform. One must also keep in mind that packet loss can effect real time data analysis. These are issues to consider when deciding on a protocol. In this section follows a discussion on the most common application layer protocols for IoT and WSNs.

2.7.1 HTTP

"The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems"[5]. HTTP is the dominating protocol for client-server application level communication in the Cloud context. However, HTTP has proven to be energy ineffective in constrained environments, in particular with the small frame sizes and the lossy links of low-power wireless communication that characterizes traffic from IoT

devices and sensor nodes [16] [18]. Therefore, HTTP is seldom used in IoT and WSNs and the following protocols are preferable.

2.7.2 CoAP

The Constrained Application Protocol (CoAP) is "a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks" [6]. It was designed by the Internet Engineering Task Force (IETF) especially for devices with constrained resources. This makes CoAP a good option to use with IoT devices as they are usually resource-constrained. IETF state that "The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation"[6]. CoAP runs over UDP in order to remove the TCP overhead and to reduce bandwidth requirements. The protocol enables RESTful communication in a client-server architecture with the well-known methods GET, PUT, POST and DELETE and can provide both synchronous and asynchronous responses creating painless interactions with HTTP in web applications [6] [16] [18].

2.7.3 MQTT

Message Queue Telemetry Transport (MQTT) is a client server publish/subscribe protocol created by IBM with M2M communication in constrained environments in mind. It runs on TCP and is asynchronous. IoT devices are usually resource constrained, making MQTT a good option to use with IoT devices. The protocols publish/subscribe configuration demands less resources than request/response as clients do not have to request updates which decreases the network bandwidth and the need for computational resources. Comparing MQTT to CoAP we see that the UDP-based CoAP produces lower overhead than the MQTT which runs over TCP. However, because of the lack of TCPs retransmission mechanisms, packet loss is more likely using CoAP and with low packet loss MQTT can experience lower delays than CoAP [16].

2.7.4 AMQP

The Advanced Message Queuing Protocol (AMQP) is ”an open standard for passing business messages between applications or organizations” [2]. AMQP originates from the financial industry has been reported to have an environment of 2000 user processing 300 million messages every day. AMQP supply asynchronous publish/subscribe communication and has a ”store-and-forward” feature that makes it reliable even after network disruptions [16].

IoT is still an emerging concept and therefore there are many standards trying to solve the same scenario.

2.8 Relationship between the NodeMCU development board and ESP8266

This is relevant background information to the reader to avoid any confusion about what the difference between the esp8266 and the NodeMCU is. The two are referred to as the same through the thesis, but it is relevant to understand that the NodeMCU is a development board that contains the esp8266. The NodeMCU is a micro controller unit that implements the Espriff esp8266 WiFi enabled chip on an open source configuration. A standalone esp8266 can be purchased. Then the user of the board would have to supply the other necessary components such as a USB to serial bus, GPIO pins etc. The NodeMCU gives an end user a pre-made board which contains the components necessary to do development without the need for soldering. Esp8266 and NodeMCU are used interchangeably through in the description of the sensor packages [12].

Chapter 3

Technology considerations

For the purposes of this thesis it is important to keep in mind some of the key consideration a cloud based monitoring and visualization system will have. The following chapter lists out the key considerations for the technology involved in the implementation of the system. These technology considerations where kept in mind through the system design phase and helped in gaining an understanding in what to consider during the system implementation.

3.1 Challenges related to throughput

Any system that is processing large amounts of data will need to be able to process these data as they arrive. In the proof of concept there are five motes sending out data every 15-30 minutes. But it is still important to keep in mind how the web-API design can be efficient at scale. As an example, if the web-API is not able to handle a high amount of concurrent requests the system could deploy multiple copies of the web-API behind a load balancing mechanism to achieve good horizontal scaling.

A wireless sensor network also have major read write challenges transmitting the data. Should the WSN motes transmit directly to the internet, or would it be more beneficial to group data in an edge node and transmit larger chunks of data. These are important things to consider as the ability to replay events and resiliency in the network is important to give a clear picture of the soil moisture in the monitored area.

3.2 Data stream management

There are a number of things to consider when looking at the streaming that is configured for this system. The main question one needs to answer when setting up a stream management system is whether or not you want to run queries on the live stream. This proof of concept does not implement analytics, notifications and real time visualization. If this was to be implemented, the system would need the ability to query the stream. One suggested approach is to use the offerings from the big cloud vendors. IoT gateway can feed data to a stream analysis service in Azure and allows for real time querying of the data before it hits the database.

The system could have used a pub/sub pattern to subscribe to data arriving in the database. There are many frameworks which allows for notifying listeners when a new object arrives in the database. For the implementation I experimented with Facebooks graphQL specification implemented in node which allows for subscriptions to sockets that publish when new data arrives. This would not yield true real time analysis, however given the requirements described in the reference architecture one can imagine that true real time is not needed as the data monitored by the system does not change rapidly.

3.3 Hosting

The way to host is a major cost driver and to a large extent decides the feasibility for implementations with similar characteristics in large parts of the world. And unfortunately we are seeing that the cloud capabilities in areas where water is the most scarce is not the best places to utilize cloud technologies [4]. The system that was built could also be hosted as an on premise application on a single network. In that case the cost of the system would be different, and could vary greatly in implementation cost based on the number of motes deployed.

When considering a cloud deployment of this system the most important things to keep in mind is the network coverage, which over most of Norway is good. However, this system could also be deployed using the 4G network as the data packet sizes are limited. A system with the same characteristics as the one described in this thesis could also cache the data from the motes on an edge node

which can act as a gateway to the cloud, or in certain cases cache the data on the mote itself given that the system uses hardware with substantial amounts of ram and flash storage. For reference the esp8266 used for this proof of concept usually ships with 4MB of flash storage [12].

3.4 Data integrity

The primary factor for enabling data analysis is the overall integrity of the data-set, as this is what will be used for analysis of water consumption. That is why the main challenge for this system and the main consideration for the system is whether a cloud based database can support the integrity the system will need to gain insight through using the data-set.

The secondary factor of data integrity relates to the connectivity of the sensors, and whether or not the system is able to transmit precise and timely data for the monitored variables. In this case this has more to do with the networking of the system and the way the motes are programmed.

The concept of eventual consistency would be followed for this type of system as you might end up with a data set distributed across multiple regions and a read and write specific database. Meaning that the system would not necessarily need an always consistent model given that outliers would be weeded out if the system normalizes the data. The missing event occurring between two data points would be from an intuitive standpoint trivial to approximate.

3.5 Scaling

Any system gathering large amounts of data would need to easily work at scale. There are a number of strategies that will be covered in the implementation notes, both for the concrete implementation and the reference architecture that is described in this thesis. One important concept is to keep state away from any interface that interacts with the database, and leverage the timestamps to do the heavy lifting of stitching the data back together.

3.6 Data quality

A key consideration is ensuring that the data is of high quality and that the system can filter out the values that deviate too much. Over time the data sets will give an impression of normal values and through analysis it is safe to assume that the system can filter out data that should not be a part of the displayed data. It is important to understand that it is easy to tell in a visualization when a data point is an unrealistic outlier, but it is better to have a version of the data set that does away with these data points entirely. One approach that could be considered is normalizing the data before the system sends it to the display component, or implement a system that scans and cleans the data. In general it is more prudent to keep the data intact, as one could regenerate the same result again as long as the underlying data of the calculation is kept intact. This could allow for experimenting with different approaches for data quality assurance.

Chapter 4

State of the art and reference projects

This chapter aims to examine reference projects that share characteristics with the proof of concept implementation in this thesis, as well as looking at commercial actors within physical sensors and dashboard software.

4.1 Technology applications in agriculture

Approaching technology in agriculture it would be possible to look at a range of different subjects. Examples could be food science, engineering innovations etc. For this thesis what is most relevant to look at is technology applied to agriculture in the sense of information technology. Below there has been compiled a list of reference projects relating to the research of this thesis that have been helpful in gauging the to what extent systems of similar characteristics have been implemented before. In addition to this the section also looks at some cutting edge commercial solutions and systems that share some characteristics with the proof of concept implementation.

4.1.1 Automated Irrigation System

J. Gutiérrez *et al.* [15] has developed an automated irrigation system using a WSN comprised of soil-moisture and temperature sensors placed in the root zone of the plants. The systems consists of senor units, gateway nodes, the

irrigation mechanisms and a web server hosting a web application. The sensor units communicated with the gateway nodes using ZigBee and the gateway nodes transmitted the data to the web server through a GPRS module via the public mobile network. The sensor units was powered by both battery power and solar power and contained the two sensors, a microcontroller unit and a ZigBee transmission unit. The gateway nodes contained a ZigBee module, a microcontroller and a GPRS module. The gateway nodes received, identified, recorded and analyzed the sensor data as well as controlling the pumps in the irrigation system.

The motivation for the system was to optimize water use for agricultural crops in water scarce environments. During the 136 day test in a sage crop field "water savings of up to 90% compared with traditional irrigation practices of the agricultural zone"[15, p. 174] was shown. J. Gutiérrez *et al.* argues that "the modular configuration of the automated irrigation system allows it to be scaled up for larger greenhouses or open fields"[15, p. 166] and that the system is "feasible and cost effective for optimizing water resources for agricultural production."[15, p. 174].

This system shares many characteristics with the proof of concept described in this thesis. However it discusses a field study and the results to a larger extent than the system implementation. There is seemingly no focus on the cost of the system and they do not describe how it can be deployed using cloud services. It is valuable input in providing an answer to the questions whether a WSN and data gathering system is a fesible way to reduce water usage in agriculture. The goal of this thesis is to go more in depth in describing the archtitectuaral and technology considerations in addition to discussing the feasability of such a system.

4.1.2 Vineyards

J. Burrell *et al.*[8] focuses on how the agricultural workforce should play a role in the shaping of WSNs. The goal is to develop a WSN for use in a vineyard. The author has a human-centric research approach and use "ethnographic methods including interviews, site tours, and observational work to broadly understand the work activities and priorities of the various roles working in a vineyard"[8, p. 38].

4.1.3 On-farm frost monitoring

F.J. Pierce *et al.* describe "hardware and software components of technologies developed for regional and on-farm sensor networks and their implementation in two agricultural applications in Washington State, an agricultural weather network and an on-farm frost monitoring network" [30, abstract] . The objective of the work was to use the technologies to improve "important farming operations that add value through improved efficiency and efficacy of targeted management practices"[30, abstract]. For the on-farm frost monitoring network the authors implement a radio based star topology telemetry network and a ad-hoc software solution to collect, manage and display the data. The authors cooperate with the farmers to determine the requirements of the system. The system needs to "report air temperature every minute to a computer operating anywhere on the farm"[30, p. 35], it was important that "multiple workers involved in the frost protection event be able to view current and trend data for each monitoring station in real time"[30, p. 36] and that "The user needed to be able to set temperature thresholds that would trigger an alarm at their computer"[30, p. 36]. Although meeting some problems with weather and the radio communications, the WSN helped the farmers make more informed decisions about what to do when the frost occurred.

4.1.4 Aquaponics

P.C.P De Silva and P.C.A De Silva has applied an IoT architecture to water quality control in an aquaponics system [34]. Aquaponics is a soil less growing method in which fertelizer is created from within the system boundaries using fish waste. In this context the food production process is treated as "a distributed industrial manufacturing process with strategic data acquisition, automated optimal control, and automated process control which is to be implemented in urban and rural areas" [34, p. 1]. The system is comprised of sensors and controlling units denoted as endpoints, an IoT cluster, a streaming analytics server and a data analytics server. The system is intended to facilitate for urban agriculture and addresses food security issues for crops produced intensively in limited spaces.

4.1.5 Summary

As can be seen from this related work there are multiple methods and designs for monitoring different faucets of agriculture. And we can also see that similar implementations have been done. These implementations have a higher focus on automation and monitoring. This work has been helpful in setting a scope for what has been achieved by others, and helped in forming the conclusions of this thesis. Especially in answering the question of whether a system with these characteristics is a feasible way to monitor and reduce water usage in agriculture. The goal is to expand on the concepts presented here by presenting both a generic design and a specific implementation as well as discussing the cost of hosting systems like this to a larger extent than what was done in the reference projects.

4.2 Off the shelf hardware and software

One revolution that has happened within sensors and sensor networks, is the high availability of general purpose small computers like the raspberry pi or the Arduino. The falling price of fit to purpose sensors like the humidity sensors utilized in the proof of concept has also changed drastically. This enables researchers and entrepreneurs to rapidly prototype and test hypothesis with hardware that used to cost hundreds of thousands for a fraction of the cost. This has fueled a plethora of easily configurable software and hardware.

When it comes to the commercial side of this type of technology, this market seem to have gotten more mature recently. And there is a lot of content about this type of technology as well as commercial implementations of similar technology.

There are examples of commercial operators in this space and in this section we will have a look at some of the operators in this space.

There are not many off the shelf hardware in so far as ready made packages found during the research for this thesis. Most of the moisture sensors that are commercially available are manual read out sensors that tell you when to water your plants. The same sensors that where used for this proof of concept comes up when searching for hardware. There are however different types of this sensor that have other capabilities than the ones used in the proof of concept. These are

generic, so I have not been able to find a spec sheet for the sensor. But it is hard to gain insight in to how they work compared to as an example the Sensorion SHT11 sensor [33] where you can read the full spec of how the sensor works without being a customer of the company. In comparison the sensor offerings from Bosch and Schneider there is no concrete technical data easily available to the general public. I acknowledge that these companies produce hardware for a different customer segment. At the same time it makes it hard to do a objective comparison of the sensors mentioned in this thesis and the commercial offerings from these companies.

4.3 Commercial actors and products

4.3.1 Inmtn

"Inmtn" is an industrial company focusing on among other things irrigation control. They have industrial scale solutions which is used for monitoring the factors that this proof of concept was able to monitor. This company utilizes a more generic type of control architecture called SCADA. SCADA architecture focuses on the entire production process instead of only the gathering of information like this thesis is focused on for the proof of concept implementation **inmtn**

4.3.2 Schneider

Another more well known company that offers products and solutions similar to Inmtn is Schneider electric. They have also built out a suit of tools for monitoring in agricultural applications. Schneider offers products that have taken the systems discussed in this thesis to a production level. As an example they offer hardware that integrates with existing controls for irrigation systems. They also offer systems for soil moisture monitoring that monitors some of the same variables discussed in this thesis. Interestingly they have opted for burying the sensors at multiple depths, and are also utilizing cloud services in the operation of their platform. Per November 2019 you can see a demo at https://scadafarm.azurewebsites.net/home [32].

4.3.3 Wildeye

Wildeye is a company that seemingly focus on smaller operations as one of their key selling points is the price of their hardware and software. Their solution is similar to what is outlined in the system implementation proposal. They have weather proof sensors which are wired to a central computer which distributes the information wordlessly. This differs from the proposal as you are not running a wireless sensors alone, but in tandem with other sensors [39].

4.3.4 Bosch

Bosch also features smart agriculture as a part of their website. It seems that this company is to a greater extent in proof of concept mode. Given that they already create advanced tools for other applications it is safe to say that they will probably launch more tools in this space [7].

4.4 Commercial actors software

These commerical actors have been chosen because they are platforms I have had exposure to through my professional work, and the fact that they are impactful examles of dashboarding and data collection software.

4.4.1 Tableu

One example of a tool that allows you to view aggregate data and build dashboards is Tebleu. This software allows for connectivity to different data sources like csv, sql, ssas, odata and more. This is a useful tool for generating views for data and is widely used within reporting in the financial sector. In this scenario one could imagine that a service that could be offered to farmers is creating dashboard views and reports using Tableu and this type of software could accelerate the insight gained from the data gathered from the sensor network. One massive drawback to this product is that the licence is more that $ 800 per year as it is heavily geared towards enterprise users [35].

4.4.2 Veracity

Veracity is another example of a visualization plattform, but this is a platform that crosses the gap of IOT and dashboard and business inteligence tools. This is a platform developed by Det Norske Veritas and is used for a range of industrial applications. Although agriculture is not yet one of the use cases mentioned on their website. This platform also has an active community that creates third party applications that can be leveraged to accelerate the insight and analysis process [38].

Chapter 5

Cost constraints

This chapter aims to give insight on the real costs related to implementing the proof of concept monitoring system described in this thesis.

5.1 Cost and context

Without getting to philosophical we need to consider the simple question: What is expensive? This is important as it sets the context for the discussions that will follow in this chapter. This section will be looking at costs in two different ways, what is the cost to society, and what is the cost measured in monetary value.

It is clear that excess water usage has negative externalities and costs to society, consider the example of drought in Norway during the summer of 2018. We still need to keep the negative effects of excess water usage in mind, this is mainly to understand why reducing water usage in agriculture is important. Depleting groundwater can also increase inequality as it can quickly turn in to an arms race where farmers drill deeper and deeper looking for water.

5.2 Considering the world wide availability of the internet

There is a discrepancy in the availability of internet connectivity in the developed and the developing world, and although this is improving over time with

adoption of 4G/5G and investment in internet and communication infrastructure. This is a major hurdle that could face the implementation of a WSN system in certain regions of the world. And unfortunately the areas that are especially drought prone can also have other challenges on the infrastructure side. The solution presented assumes a certain level of development within telecommunications and internet infrastructure that is not available through the entire world.

5.3 Cost of hardware

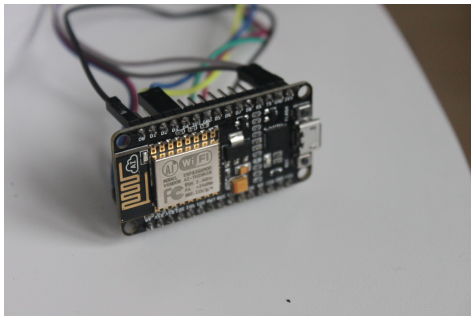
Price examples from Alibaba (e-commerce platform) per November 2019

Component	Price	purpose
NodeMCU	$ 2.35	microcontroller
Generic moisture sensor	$ 0.50	sensing
STH11	$ 12	sensing
3v 1200 mah battery	$ 3	power

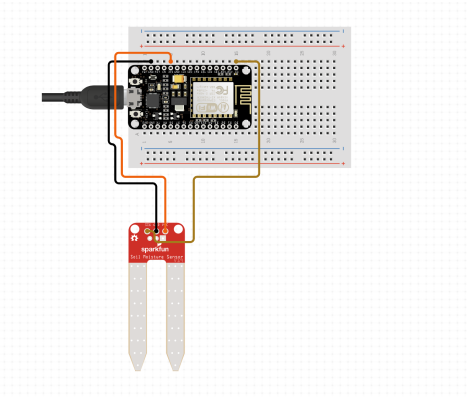
Depending on the version of the sensor that is chosen one can see that the impact to the price of the sensor package greatly. The information from the generic sensor that measures Resistance in the soul is interesting enough to warrant building a package containing just that sensor. The total cost where similar for a portion of the packages that where built later in the project as more of the components where ordered online. In the initial part of the project the components where purchased through a Norwegian retailer at approximately 3x the price. In the end the listed prices based on the retailer Alibaba gives a more realistic picture of what the cost of implementing one sensor package would be. All in the deployed sensor packages where built for $ 5.75 with a battery and approximately $ 2.75 without a battery. Which was incidental the configuration deployed for the proof of concept.

5.4 Cost of hosting

The cost of hosting is highly linked to the architecture that the application is designed around. Consider the following, redundancy, managed services and hardware needs. All of these factors will play in to what the total cost of the



(a) Nodemcu



(b) Nodemcu schematic

Figure 5.1: Draft and real world version of the sensor package

system will be. To keep things simple and realistic for the purposes of evaluating realistic costs for the proof of concept system will adhere to a three tiered architecture with all the components hosted on the virtual machine types listed in the table below. Following is a short review of two different scenarios using IaaS and PaaS. For IaaS it is assumed that one server is sufficient to host database, backend and frontend. For PaaS it is assumed that it is sufficiant to use one hosted web app container and one hosted document db solution. For the PaaS it is most applicable to review AWS and Azure as Digitalocean has not been a major platform for PaaS.

Provider	VM INSTANCE	Price/month
Digitalocean	small droplet	$ 6.25
Azure	linux vm (b1s instance type)	$ 7.6
AWS	ec2 t3.micro	$ 6.86

Given the architecture that was implemented, the cost breakdown would have been the following for the three hosting platforms that where tested. One reflection is the fact that AWS and Azure offer more ancillary services and would probably have made future imporvements to the platform easier to implement. As an example there are excellent packages for application monitoring and usage statistics in both AWS (cloudwatch) and Azure (application insight) which are not present on Digitalocean. So the total value of the Azure and AWS platforms are increased by the boundary resources that are offered by the platforms. In a production system the increased price of AWS and Azure could be worth it as

the service ecosystem is more mature.

Although the IaaS deployment model described above was what ended up being implemented. It was still relevant to look at the service offerings from especially Azure. This is a valuable way of setting up the architecture of the application. One of the main benefits is the ease of deployments. This is again where the boundry resources of the platform giants are outshining IaaS provider Digitalocean. As an example Azure offers what at the time was called VSTS (visual studio team services) which allowed for simple and straight forward CI/CD (continous integration/continious delivery) which is a fully automated way of deploying code to a production environment. This is another case of a platform having its perceived value increased by offering ancillary services to the end user of the platform.

The conclusion when comparing these platforms is that it is hard to decide what to use based on price alone. Like with any product, there is a whole host of things to consider.

5.5 Deployment context

It has been mentioned in the thesis that there is a difference between a proof of concept project and a deployment context. What is meant by this is that the costs discussed are the ones that where experienced during the implementation phase of this proof of concept. What was observed is that the cost of the system grew in a linear fashion for the motes, and in a tiered manner for the server costs. This is highly dependent on what cloud provider you choose but the following would be true for all of them. The important first step is to establish a baseline consumption of computing power. In the case of the proof of concept implementation, the cheapest server available was chosen and that proved sufficient for the full implementation. Normally one would scale up by adding more machines to the pool of workers in the application. When you run on dedicated servers you would see that there is not a linear relationship between the amount of motes and the cost of hosting. Normally a threshold for capacity would be hit, and then more hardware would be provisioned based on the increased need.

If a service from one of the major cloud providers had been used for web-API

management and databases the cost picture would have behaved completely differently as there is a high chance that the number of requests and amount of data stored would have more to say for the cost of operations of the system.

To deploy this in a production setting you would be looking at a set of other cost drivers which are not accounted for in this proof of concept. But from a hosting perspective it would probably look mostly the same. It would be beneficial to leverage more hosted services as usually the bigger cost driver would be paying the engineers that are creating the application, which was not a factor in this implementation. At that point it would make sense to take a serious look at the different hosting models for cloud.

5.6 Summary

As seen it is important to scope and explain what "expensive" is based on the deployment context. Given that the thesis researched the implementation in a Norwegian context, with a Norwegian cost level, the following is observed: The hardware cost is experiencing good synergies from open sourcing and cheap shipping from China. The hosting cost are being pushed down by a highly competitive marketplace where providers offer good rates for computing power. The software packages that where used in this proof of concept where open source and contributed greatly in accelerating the implementation of the proof of concept. As far as achieving the requirement of gathering and storing the data in an inexpensive manner, the cost break down shows that the system outlined in the proof of concept could be implemented at a low monthly rate.

Chapter 6

Overview of a proposed implementation

This chapter describes and evaluates the system implementation. It describes all the activities that went in to building the proof of concept. It covers the initial design plans, and a conceptual deployment of a system based on those plans.

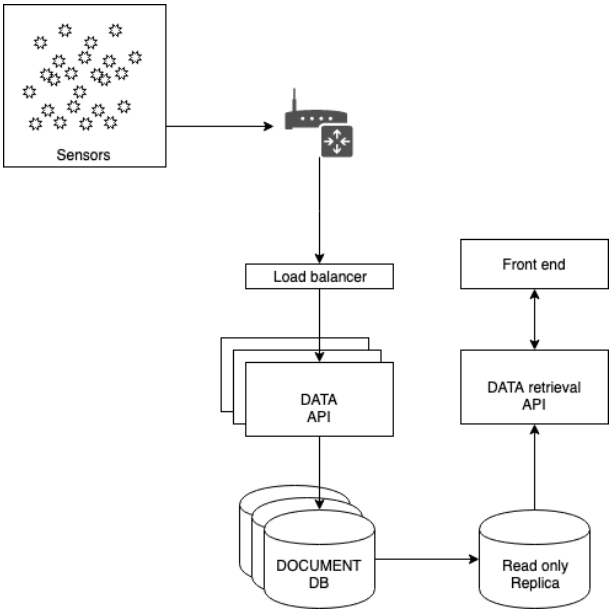
6.1 Describing an ideal system design

Early on I had a clear vision on how to build a system that scales well when the amount of traffic increases. And that the system can handle multiple user sessions without impacting the performance of the data gathering APIs. Figure 6.1 is an illustration of the reference design that was used for building the proof of concept system. This diagram omits any gateway or edge node in favour of a system where it is assumed that there is good networking coverage and that the motes can connect directly to the internet.

6.2 Load balancer design

A load-balancer would be responsible for unifying all of the stateless apis and exposing them as a single DNS address. As an example, a website like exam-

Figure 6.1: Reference system design



ple.com points to a single machine/ip. The same address could point to a load balancing server/service that divides the load between a number of the stateless web-API. A load balancer typically works on a algorithm to assign requests to any of the APIs in the system. A common example of this type of algorithm is round robin. This would be a good strategy for handling load increases as more and more instances of the stateless APIs can be added to the load balancers list of available workers.

6.3 Web-API design

For this system a web-API first approach was used. In an web-API first approach it is important to think about the resources that is needed in the web-API. As an example, the web-API design is approached by breaking down the model of the objects that would need to be contained within the system as this quickly can give a good impression of the resources that the system will need to create, read, update and delete.

Generally, web-API design has a clear segregation between resources by implementing controllers that have very specific jobs. This is a design pattern that

lends it self to document based storage solutions as the objects and resources that are accessible through the web-API, are typically similar to the schema of the database. A good web-API design will lay the foundation for meeting the reqirement of the system being able to handle a large amount of data from a sensor network. Web APIs are built to handle a large number of concurrent requests at random intervals, such as retrieving data for displaying products or news to users. This is valuable as the requirement for the system is to handle a large amount of concurrent requests. To ensure that any request can be processed regardless of data arrival order, the design of the web-API is stateless. A general observation is that this is easy to achieve when the web-API is responsible for creating entries and these entries are timestamped. This allows for offloading the sorting of the data to the client, and ensures that each request will be self contained.

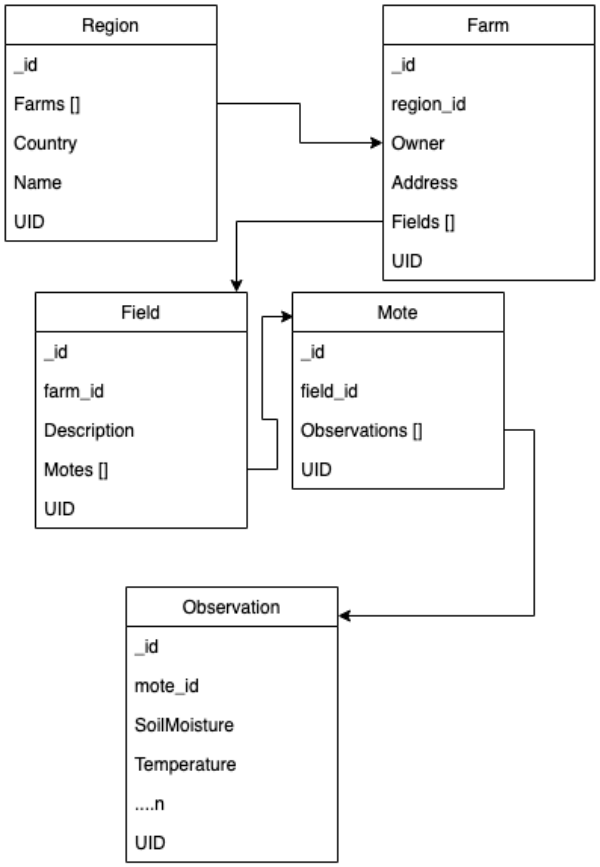
6.4 Database design

The database design is often based around the structure the data. And in the case of the proof of concept the system has a clear definition of what data points to gather and what the model for said objects would be. One thing to note is that through this proof of concept it became apparent that the model for this system is one that is a natural candidate for nested objects instead of a relational database. The implementation idea was to have a high performance implementation of MongoDB, and if the need presented itsels long term storage could be added, and high performance caching to the read web-API. Another important concept of the design is the read only replica of the database to be used for the view portion of the application.

6.5 Model design

The model design says something about the relationship between the different objects in the system and how they relate to each other. The focus here is taking in a view of the whole system, the goal is a system that could scale beyond one single monitoring location. The design also takes in to account how the user can access data from a farm level, or if the route is chosen, how interested parties can access the data from a region level. The proposed model design could ease the

Figure 6.2: Model design



analysis of aggregate data, and at the same time provide the foundation for a fast and reliable way to update the database.

While document databases are a good way to nest data in a manner that is logical given their relation, it is still important to keep in mind what parts of the model could potentially slow down an insert if it needs to go through a lot of steps to reach its destination. As an example *farm.mote.observation* is a good way of finding data when you are in the context of a view. Considering the insert, it will be more beneficial to log an observation on the mote level, instead of considering what farm or area the observation belongs to first. This design challenge in the model could be solved by saving references to the different relevant IDs instead of embedding documents. In the web-API layer there has emerged technologies that abstracts away the embedding and allows the user to select whether they

want to see the nested version of an object. GraphQL allows users to submit a schema that they are interested in, and as long as the web-API has definitions for where to look for the referenced objects it can pass back the full embedded document/nested object to the user. This achieves the same positive effects as the embedding, without the drawbacks of intermingling the objects.

6.6 Implementation conceptual

Cloud technology has enabled implementation of large scale systems at a much lower upfront cost, this is advantageous in monitoring and ingesting large amounts of data.

There are three main components that need to be considered and the first and most important is how the motes are linked to the cloud. There are a multitude of concerns that would need to be addressed like potential bottlenecks, security, and choosing the correct gateway for the motes, however most of this is considered to be out of scope for this thesis.

The architecture allows the motes in the system to be highly autonomous/ low coupling and no dependency to each other. This is a choice that requires more bandwidth usage, and is most advantageous when there are no caps, and the internet connection is not cellular.

The point is to have every mote operating individually and having an isolated connection to the ingestion web-API. This means that the motes have their own connection to a router and is free to communicate with the internet. This raises some challenges regarding security as a larger part of the sensor network is internet facing. This is something that would have to be treated carefully and given more thought in a production environment. This layer of the architecture also describes the network layout and the structure is proposed for a multi site setup for precision farming as well as the network structure for the proof of concept environment.

Each mote is connected to the internet via a normal WiFi router. Every mote has an ip address, but you are not able to give the proof of concept motes remote instructions. Changes have to be made by flashing new software to the chip.

The motes acts just like any client on the internet and is able to send post/get requests just like a normal computer. This means that it is easy to program the motes for any developer with basic web programming knowledge. The computing power requirements for the motes are low as the system is intended to read data at a low frequency (test data collected at 30m intervals).

The mote network fits the description of a star network where the common connection point is the router that gives access to the internet. There are multiple configurations that are relevant in an IoT network, and star is one of the more rudimentary topologies. In this case this is what makes the trial environment easy to implement.

Ingestion is an important component given that it would need to scale when more monitoring sites are introduced to the system. A simple ingestion engine has been designed. The ingestion is a one to many relationship and the data has a single point of entry. This is the same in both Azure and the custom ingestion engine. There are many things to consider when creating the infrastructure to handle vast amounts of data, and the entire stack of the technology needs to be optimized for large amounts of data.

When the data has traveled from the deployment site to the cloud the system is dependent on reliable and cost effective storage of the data. Given that many IoT devices uses Javascript object notation (JSON) to communicate with a server a natural choice for storing data is a JSON based document database like MongoDB. This type of database is capable of many of read and writes per second. Due to the nature of the application write performance is the most important. One big advantage of cloud based data base systems is that they often do automatic replication over multiple locations something witch is great for data redundancy.

6.7 Site deployment conceptual

Following is a conceptual site deployment for monitoring a grass field of approximately 9ha, which incidentally happens to be the grass field of my farm in Hedmark. The idea is two think of this farm as having two "fields" and both fields being of interest. Why do I want to split these fields? I have observed that

Figure 6.3: Site deployment

the yield is significantly different as one of these fields get a lot more sunlight than the other. It will also be of interest to describe the networks requirements over such a spread out area.

Field 1 is the larger of the two at approximately 8ha while field B is approximately 1.5ha. To cover this entire area with wireless signal you will need to implement a network that can hand over connectivity in a seamless manner. We could go many different routes for the configuration. I theorize that the easiest way to implement it is using repeaters for the signals. I would probably need to run power to some strategic locations in the field. Assuming a modest range of 150m I could in theory cover the field by using two WiFi access points, and with amplification we could cover both fields. Just from this exercise we see that networking could be a hurdle. However this would not be a stopper for gathering data as there are alternate ways of communicating via radio signal.

When looking at the placement of the sensors I would consider the elevation, and the sun conditions in the patch the sensors are being deployed, seeing how much the reading vary would be hugely beneficial as it could tell a lot about the amount of sensors I would need to deploy to get satisfactory readings. As well as some of the different areas of the field being more prone to retaining water and having different soil qualities.

Chapter 7

Proof of concept for monitoring soil moisture

This chapter will cover the proof of concept implementation for both hardware and software. It will describe the designs and concepts explored in the thesis as used in a real world setting.

7.1 Components of the proof of concept

The proof of concept consists of a simulated Wireless Sensor Network(WSN), a real life small scale WSN and a cloud hosted web application. The simulated WSN is a representation of a general network of sensor nodes measuring soil data in an urban agricultural setting. The simulated WSN is based on data from a real life small scale WSN implemented. This real life WSN consisted of nodes based on the NodeMcu development board which collected soil data from the houseplants. The data was stored on a web server and later used to create datasets for the simulation. The WSN was active and collected data for several months. The goal of the simulation is to test what kind of hosting service is nessecary.

The solution is a testament to how minimalistic the application and interface part of systems with similar characteristics could be done. The application is hosted on an affordable cloud service. This shows that this system can easily handle the traffic a WSN of this size produces. Based on datasets produced in 2017 the simulation can scale the WSN to stress test the web application.

7.2 NodeMcu

The sensor nodes are based around the NodeMcu development board. The NodeMcu board is again build around the ESP8266 [12] WiFi chip. To develop an application for a NodeMcu board, you must first create the firmware for the ESP2366 chip. The chip comes with standard firmware, but for software development you need more libraries than the standard firmware. The firmware is modular and you can put together custom firmware using an online firmware build service called nodemcu-build.com. When you have flashed the firmware, you can write Lua scripts which you again flash to the chip. For the application, only simple scripts are needed. For the nodes the scripts reads sensor data from the analog and digital input pins on the board, creates a data-point and a HTTP request and sends the data to the server. This happens every thrity minutes. When the request is sent, the device is set to sleep mode to save battery/power. The boards needs 3v power to run and can use both batteries and normal power.

It is hard to run this type of board on battery power as there are quite many of the components that consume a lot of power. The esp8266 controller has a built in low power sleep mode which can be utilized for ultra low power operation. The problem for the proof of concept is that the board that the NodeMCU controller has many additional components like the USB (universal serial bus) controller mentioned above. This halted the progress for creating long running nodes in the system. The battery powered mote was running for up to five days in battery power with the NodeMCU board, proving that creating a completely wireless system is possible. The esp8266 board was purchased without the additional components of the nodemcu board, but completing the electrical component work proved to be more challenging than first thought. And it was decided on leaving that part of the project behind to focus more on the software and proposed architecture of this type of system.

7.3 Creating data-sets

Data visualization is an important step in showing the power of the data that the system gathered in the proof of concept. It illustrates that it is possible to gather data the gives a good representation of the current state of a plant/patch of land. As you can see from the illustration underneath this data is intuitive.

For the proof of concept two different soil moisture sensors where used. One is a rudimentary sensor that measures the resistance in the soil between two metal rods. This gives a good indication of moisture level. Water is a good conductor of electricity which means that a reduction in resistance means that there is more water present in the soil

The second sensor implemented (SHT11) is a humidity sensor paired with a special housing that allows for air to pass through, but not water. This means that as the soil moisture increases the moisture of the air that passes through the sensor. This type of sensor also contains a temperature sensor, which gives valuable information of the current state of the soil that is being monitored.

For the proof of concept the sensitivity of the sensors was not measured. However, to establish a usable humidity readout a span between the readout from 100 percent humidity (sensor submerged in water) and as close to completely dry soil was created. For the humidity style sensor a baseline verification was established by checking that the sensor was within a few percentage points of the humidity reported by "metrologisk institutt" outdoor in the area for the testing of the sensors.

The resistance based sensor is analogue, and it is easy to read the information from the sensor. As it is connected to a pin on the board that is able to read resistance. The humidity sensor is digital and required that a library for communicating with the sensor was loaded on to the micro-controller. This type of open source software is often easy to find online for most types of micro-controllers. This also held true for the micro-controller that was used for the proof of concept (NodeMCU). A Lua implementation that gave a good read out to establish a baseline for temperature and humidity was also found and leveraged for the proof of concept.

The data collected was initially limited to only moisture, this was interesting as the data shows the soil moisture over time. A challanging part of the beginning of the proof of concept was figuring out what thresholds should be set for watering. Early on realizing that the system was ignoring factors like evaporation, the possibility of also monitoring the temperature of the soil in addition to the moisture level. This was also gathered as data for the system and proved

visually interesting, but was unfortunately never used for any analysis of correlation between temperature and rate of moisture reduction.

Ultimately I realized that I had taken an oversimplified view of what is important to look at when you try to understand the complex picture of plant health. I also realize that water is the primary factor that we would like to monitor, and I saw that the data I was able to capture gave a good picture of the water consumption and I do theorize that this data can do a lot to cut down on water usage in agriculture.

7.4 Web Application

The web application is a simple application built on the MEAN-Stack. The application receives and stores the data points from the WSN motes while providing a web interface. The web interface shows one graph for each area in the WSN. The graphs show the average temperature and soil moisture in that area for the last 24 hours. Each point on the graph is a calculation of the average temperature or soil moisture from all the motes. The interface is chosen not for actual use but as a proof that the web application can handle a certain traffic and functionality.

7.5 MEAN-stack

The MEAN-stack is a combination of technologies that allows developers to create lightweight web application written in all Javascript. It consists of the four elements MongoDB, Express.js, AngularJS and Node.js. MongoDB is a noSQL database, Express.js is a server side javascript framework, AngularJS is a frontend javascript framework and Node.js is a server side Javascript runtime environment.

7.6 Database

For the proof of concept it was decided to go with a noSQL system. This is different from a regular relational database as the stored files are can handle nested object, instead of using a key relation as in a relational database. This type of

system lends itself well to nested objects, as there is a clear relationship and tree like structure to the layout of the data model. The data model was only concerned with storing the moisture data, and not the kind of administrative data that you might need for this type of system in the real world

All calculations and data aggregation are done in the browser. This is done to offload the server, as the server should only handle the HTTP requests. The performance of the browser side application is decent, however the Javascript library used to generate the graphs have a lot of functionality, like animation that are not really necessary for this purpose, which gives a sub optimal rendering time. For this situation, a simpler library would be better, however the point of the web application is not to prove response time of Javascript libraries, the purpose is to prove the responsiveness of the server. It is also relevant to point out that using AngularJS as the main front end framework is not the best fit for this application. For rendering of this kind of front end components (graphs) React in combination with other libraries would be a better choice as it can have higher performance.

7.7 Hosting

This proof of concept implementation looked at the following vendors: Aws, Azure and Digitalocean. The two former are the hosting giants Amazon and Microsofts offerings and is what is currently commercially viable for larger companies. And Digitalocean is a IaaS offering that is geared more towards the customers who are fine doing their own patching and upgrading. The three platforms also differ a lot in that AWS and Azure offers a large range of software as a service, and Digitalocean only offers infrastructure as a service. For digital ocean this has changed in recent times as they have started to offer hosted mySQL (open source relational database) databases, but did not at the time of implementation.

The application is hosted on a Digital Ocean Droplet. Digital Ocean is a cloud infrastructure company that offers affordable cloud services. A Droplet is a virtual machine instance with varying degrees of computing power, storage and networking functionality. The application is hosted on the cheapest possible option. A Droplet with 1GB memory and 25GB storage. This is at a cost of

5 dollars per month, which is an insignificant expense relative to the price of purchasing the hardware up front (VAT/moms was introduced in 2018).

Often there is no need for more than a simple server to run this type of application. Efforts where made to explore alternatives when it comes to cloud hosting. This proof of concept has cost as one of the key constraints, as this system would need to run at a as low as possible cost so that the investment in implementing the system gives back quickly. Seeing how we can get away with five us dollars per month (6.25 after vat was added in 2018) on digital ocean, makes it hard to compete for the other providers in our case. However it is still relevant as I need to take in to account the maintainability and scalability of our solution.

7.8 Simulating WSN

The solution uses a python script to simulate a large wireless sensor network (WSN). The script can simulate a data stream from variable sized WSNs with any network topology. For the solution, a large WSN was simulated, where the individual motes communicate directly with the main application. The simulated WSN was organized into virtual physical areas, each area containing 50 motes. This way the system can easily scale the simulation for stress testing by changing the number of areas. It's important to note that a really accurate simulation of a WSN not possible with a script like this as it can not emit all HTTP requests simultaneously.

The simulation was used to both simulate normal WSN behaviour and to stress test the web application. For normal behaviour the system simulated a WSN with 500 motes, each emitting momentary temperature and moisture data every 30 minutes. For stress testing the system had the script constantly emitting data points from all motes. These are some notes on what was experienced:

Normal WSN behaviour: In normal behaviour 500 motes are emitting momentary temperature and moisture data every 30 minutes. This means the web application has to handle 500 HTTP post requests each 30 minutes.

Stress test: For stress testing the script constantly emitted data points from all

motes. The main thing to confirm with a stress test was that the hosting service could handle that much traffic, without affecting the performance of the web application. Keep in mind that the system used the lowest spec'd virtual machines from one of the cheapest IaaS providers. If this kind of hosting can handle traffic from a large WSN, it will virtually remove the cost of hosting a the web application for a WSN like the one implemented in this proof of concept. The application had no problems handling the traffic from the stress test.

7.9 Hardware implementation for monitoring soil moisture

The harware implementation was by far the most challanging part of the work, and a lot of time was spent exploring different options for sensor and micro controller hardware in the begining of the work. However it was the interest in exploring sensors and the link between computer software and the real world that sparked the topic of discussion for this thesis. A short timeline on what was tried will be outlined, then concluded with how the deployment ended up.

7.9.1 Summer 2016

As a hobby project during my final year of college I started looking a lot at the RaspberryPI and the GPIO (general purpose input output) pins of the microcontroller and the different use cases. I had become inspired by a smart plant project I found on a Microsoft blog to try and create something similar myself. The first part was finding a low power microcontroller that had similar capabilities to the RaspberryPI and could be used with the same sensors. This is the time I learned about the NodeMcu implementation of the esp8266 chip that has been mentioned throughout this thesis. Substantial time went in to researching the different options that came along with the microcontroller. There are different micro-os'es available for the nodemcu and I settled on implementing the one that is bundled with the product. There is also a micropython implementation avaialble but I found the community resources around this os implementation to be smaller than the lua based nodemcu package that came pre loaded.

7.9.2 Fall 2016

In the fall of 2016 the majority of the time was spent tweaking the package deployed to the nodemcu. The way the OS is built up is by including different packages. As an example, HTTP capabilities would not work without including the HTTP/WiFi packages. To add on to that you can not choose to include all the packages as it can result in a build that takes up to much of the internal memory and results in an unstable os deployment. A major challange was when the mode of the GPIO pin used to read the moisture level on the resistance based sensor. This turned out to be a mode flag for the ADC method that reads the pin value, and could be fixed by modifying the binary deployment of the OS. During this time I also experimented a lot with different enclosures and battery configurations, ultimately it was decided it was better to continue the project with USB powered motes given that the max lifespan achieved on battery was approximately 5 days.

The cause of these battery problems was that the power regulator that was bundled on the all in one implementations of the esp8266 was drawing to much power, and I would have to build my own low power implementation to get this working to a satisfactory level.

7.9.3 Spring 2018

Testing of new sensors indoors happened through 2018, this was a new way of reading data as the new sensors (SHT11 [33]) that where ordered used a digital system for communicating its values. This was where the STH11 library was added, somebody had already implemented what was needed to use this sensor with the NodeMcu. This sensor is most widely used with Arduino boards and has better community resources built up around it. Ultimately I found that I was burning through far to much time focusing on the hardware implementation, and decided to focus more on refining the architecture and testing the system through the simulation described earlier in the thesis. Although it was costly both in time and money, a lot was learned by implementing this sensor, and it also gave good readings on the temperature of the soil. However I was never able to calibrate the moisture sensor SHT11 to a satisfactory level, and decided to return to the resistance based sensors, while still recording the soil temperature using the sth11 sensor.

7.9.4 Conclusion on hardware implementation

Ultimately some mistakes where made in getting too focused on the hardware portion of this proof of concept given that this was not the primary subject of research. However it lay the foundation for the answers that are given in this thesis given that the concepts that are discussed where implemented in the proof of concept. The implementation gave a lot of insight to the costs of deploying such a system. And is an important factor in the conclusion and answers to the questions posed by this thesis.

7.10 Software implementation sensor packages

Before you can start developing on the nodemcu firmware of the esp8266 there are a number of requirements that needs to be met.

7.10.1 Building the firmware

The team that maintains the nodemcu firmware also develops tools for flashing the firmware to the esp8266. This tool is downloaded and used with ease from both widows and unix based systems (windows was used for this proof of concept). They also offer cloud builders that will include the packages that you will need for developing the desired functionality. This is through an intuitive UI and they will acrually email you a finished binary version of the firmware which you can in turn flash on to the chip. All of these offerings are well documented and well maintained.

7.10.2 Building the software

When the firmware is loaded, you can communicate with the Nodemcu via USB as the all in one implementaion of the esp8266 used has a USB to serial chip. There is a IDE you can use to send code to the board to be executed live, or you can upload files. This is similar to an interactive terminal like a developer would be used to in something like python. The system was primarily developed by using the IDE for prototyping and then implementing the same code as a file to be run by the init.lua file (see reference in appendix).

7.10.3 WiFi

One of the drawbacks of this style of development is that you would physically have to flash the motes when something in the system changes. As an example if the name or password of a WiFi connection where to change the motes would have to be flashed again. This is why in a production implementation a dedicated WiFi network dedicated to this sensor network is preferable. The code for connecting to the WiFi is one of the steps the system will retry without putting the sensor to sleep, as this is vital for communicating the data point of the sensor.

7.10.4 Reading data

The system had two different versions of the main.lua code for reading the moisture of the sensor. One for the initial sensor that the system used (resistance based) where the built in functions for reading the resistance of the adc pin of the GPIO of the nodemcu board was used. And the second implementation leveraged the sht11 library (see reference in appendix). It was found that both of these read methods worked consistently.

7.11 Software implementation server

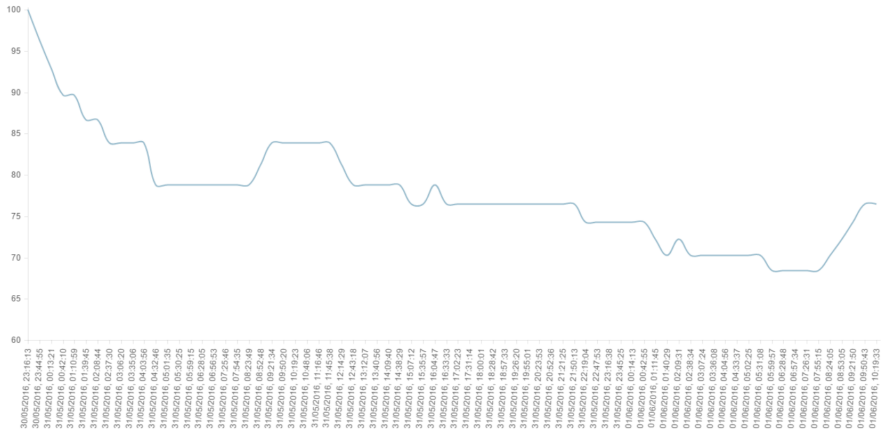
The server for this implementation was a highly simplified version of what would be needed in a production environment as the system was lacking identity and access management, authentication and logging. The programming language Node was chosen as the programming language as it has a good track record when it comes to developing restful web-APIs as well as having good experience with the language. There are also a wealth of community resources and libraries available for the language. The reason to implement it like this is that it is relatively trivial to implement a statless-web-API, which is critical to achieve good horizontal scaling of the edge resources of the proposed data collection platform.

7.12 Software implementation front-end

The frontend implementation is all about ease of use, the popularity of frameworks has changed quite a lot since 2016 with the front end framework React

coming out as the dominant framework for front end development. However the arcitechture of this application allows for changing out the front end, so even though the front end implementation is quite outdated today, an alternate front end framework could re-use the same web-API calls and javasscript packages used for displaying the data. This is also something that was given a lot of thought during the development of this proof of concept and is why a web-API first approach was found to be so beneficial.

Figure 7.1: Soil moisture graph



7.13 Cloud deployment of software

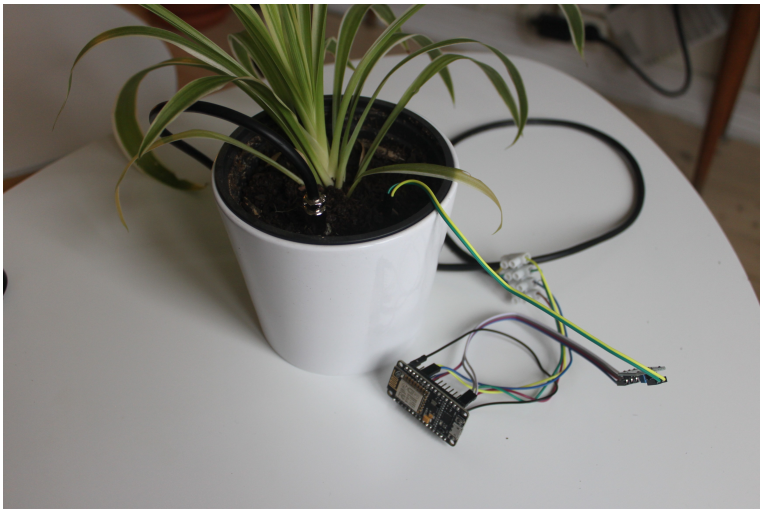
A range of options for cloud where looked at, and it is covered in the cost constraint portion of the thesis. The system had multiple iterations of the cloud deployment. The closest to a continuous deployment/continuous integration the system came to was when an implementation of the front-end was auto deployed to Azure using Azure dev-ops and visual studio team services to create a hook in to the github library used for the project. However the actual long running implementation focused around much more manual processes on Digitalocean. The system had the database, server code and front end being served from a Ubuntu server. This allowed for flexibility as well as a low cost per month (5usd + vat). Hosted applications like "meteorDB" and Azure web apps where also tested, but ultimately found that running a Linux server was easier for a proof of concept project.

7.14 On premise deployment of software

Deployment on premise is also a good option if an area has limited coverage of internet connectivity. In fact the initial version of the proof of concept ran on an on premise deployment on my personal computer. This was also an easy and manageable way of setting up the hosting. However you are responsible for more of the deployment stack yourself. Ensuring proper networking and firewall settings is more challenging than with a cloud deployment, and normally you are not leveraging pre-built images with the necessary development tools. Although it is not necessarily more expensive to host this proof of concept on premise, it does have a higher up front cost as you would need to own hardware with sufficient resources to run the proof of concept.

7.15 Site deployment of hardware

Figure 7.2: Site deployment 3. iteration



When I implemented the proof of concept I started off with an outdoor deployment in a greenhouse as the original idea was to also control irrigation using a threshold that would be discovered by the data. Because of this I had additional components in the system like a motor controller to run a pump as well as a sonic sensor to monitor the level of the tank. Although this was an interesting exercise in itself I quickly realized that although fun, controlling the

full system would give limited value to the project.

It is worth noting that I was able to run the system in the greenhouse for up to five days on battery power, and I started to see some of the challanges that would face the sensors outside in the field. One example of unexpected observation was when I noticed false readings when the sensor got heated by sunlight.

In the final implementation the same type of sensor package was deployed to five indoor plants. Data was collected for 2-3 months in the fall of 2018 and this data created the grounds for the simulation program that was written to stress test the application.

7.16 Closing remarks on implementation

The questions introduced in this thesis could have been answered from a theoretical point of view. However the implementation of the system outlined has will be of great help in answering the questions in the conclusion of this thesis. Implementing the system helped in showing how machines can be programmed to interact with the physical world. And it also helped in verifying the functional and non functional requirements set for the system, as well as testing the design theories put forth in the design chapter.

Chapter 8

Testing and evaluation of the proof of concept

Following the implementation the natural way to split up the different tests is in the order that the different parts of the proof of concept where implemented. The goal for the tests of the system was to understand how the proof of concept helped in meeting the requirements set for the system in the introduction chapter. It can be seen through the tests how each of the requirements where met to a sufficient standard, and what improvements that had to be made to make it meet the desired characteristics laid out in the introduction of this thesis. To reiterate on those requirements we note the following, the system should be able to gather and persist data from a sensor network at a low cost, it will need to be able to handle a high number of concurrent requests, and it needs to visualize the data in an efficient manner.

8.1 Web-API testing

8.1.1 description

Initial web-API testing was done on a local deployment using a web server running the web-API described in the implementation. The web-API had support for creating data points in the database, and retrieving data from said database, for display by a view component at a later stage. Through this the foundation for the platform that both the visualization component, and the sensor network

would use to communicate with the database. The web-API was tested using a web-API test tool (postman [31]). Through this it was verified that the web-API was behaving as expected for manual testing of one mote in the network. This can be seen as a manual simulation of a mote.

8.1.2 Learning outcomes

Through this testing it was verified that the design approach web-API first worked well, and it could be confirmed that data could be sent and retrieved from the web-API. Through this testing it was also verified that the database chosen was a suitable choice for this proof of concept. It also gave good insight to what steps needed to be taken to allow communication to the web-API from a local environment. However this does not recreate the conditions that need to be met for the motes to be allowed to send data to the web-API in a cloud setting.

8.2 Outdoor deployment

8.2.1 Description

Through the initial out door deployment the sensor packages described in the sensor hardware implementation where tested. This testing primarily revolved around the power delivery mechanisem as described in the implementation. It also tested the networking set up for a local on premise deployment as well as confirming the fact that the system is also able to recieve data from the motes and not just a client simulating a mote. It also tested the deployment model for the software to the motes. It tested physical characteristics such as power delivery from battery, and power delivery from a continous source. It tested the housing and last it tested that the sensors gave data that was in line with the expectation of being able to output soil moisture as a percentage from 0-100.

8.2.2 Learning outcomes

The main observation from this deployment was that it is time consuming to build a new product, even when working with good building blocks such as the NodeMCU. The product design aspects of the housing, wiring and type of sensor used are exposed to more challanges in an out door setting. It was possible to

achieve a full out door deployment, and the sensor packages where not damaged by rain in the 3-4 week period they sat outside. From a resource consumption perspective, continuing to work on the sensor packages would have required a substantial amount of time. And this would have changed the characteristics and focus of the thesis. This experimentation was useful in setting the project focus on the design, technology considerations and cloud challanges. Thus aligning the work with the questions set forth in the introduction. Given that it was chosen to not continue the work on the sensor packages from a product design point of view, and rather focus on an in door deployment.

8.3 Indoor deployment leveraging cloud technology

8.3.1 Description

Off the back of the outdoor test, the main goal was to focus the testing to the system characteristics of the proof of concept. For this it was important to reduce the focus on the sensor housing, and get back to working on the deployment model for the system, and monitoring the data in a more controlled environment. The deployment described in the implementation contained five motes, tracking the soil moisture of different house plants. At this stage the system had been migrated to the cloud platform. This is important to note as a goal for this stage of the work was to confirm that the design for the system would also work in a cloud environment. It was also important to lay the foundation for the simulation component of the system as the data generated in these months where used to generate the simulation.

8.3.2 Learning outcomes

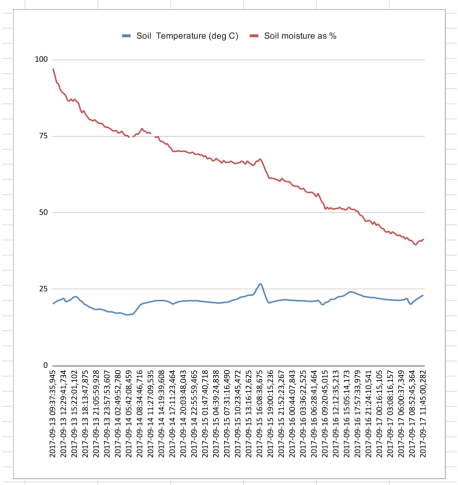
The indoor test was an important milestone as it confirmed that the design of the system was not a limitation to run the proof of concept in a cloud environment. It confirmed that the database chosen would not cause issues with data consistency as it was seen in the visualization that the data transmitted to the web-API was present and that the data set was intact. It also proved that it is possible to build multiple sensors at a low cost, for this portion of the implementation

a total of 7 NodeMCU sensor packages where built using components ordered through Ebay and Alibaba. It also tested that the system design could support tracking more variables with limited extra development required as the system was expanded to also gather temperature data in this deployment. See sample data below.


```
{"_id":{"$oid":"59b8f5b2598801622e000152"},"time":{"$date":"2017-09-13T09:09:06.226Z
    "},"soilTemperature":"19.51","soilMoisture":"79.280821917808","chipcode":"
    14790735","__v":0}
{"_id":{"$oid":"59b8fc5f598801622e000153"},"time":{"$date":"2017-09-13T09:37:35.945Z
    "},"soilTemperature":"20.18","soilMoisture":"97.08904109589","chipcode":"
    14790735","__v":0}
{"_id":{"$oid":"59b90312598801622e000154"},"time":{"$date":"2017-09-13T10:06:10.451Z
    "},"soilTemperature":"20.6","soilMoisture":"95.034246575342","chipcode":"
    14790735","__v":0}
```


Below is a graph showing moisture and temperature in the same view. This was interesting too see because the assumption was that there would be a high correlation between moisture and temperature in the soil. But from this data it is clear that the correlation is low (0.40). Although this is calculated from a small set (n = 206) it is interesting to see that the initial assumption appears to be wrong. However the test of correlation is inconclusive as it is calculated for to low of a sample.

Figure 8.1: Soil moisture and soil temperature



8.4 Simulation

8.4.1 Description

The simulation involved a new component calling the web-API to verify that the system could handle a large number of motes sending concurrent requests. From a cost perspective it was not feasible to test with physical sensors, although the price is low per sensor, not much value would have been provided by building 2-300 sensors. For this reason it was decided to build a simulation to test the systems ability to handle a large number of concurrent requests. The cloud deployment was tested with approximately 500 virtual motes sending data at the some interval as the psychical sensors (30 minutes).

8.4.2 Learning outcomes

At this stage in the testing the results had shown that it is possible to build the proof of concept both from the hardware and the software perspective. However it was not yet verified that the system could handle scale. Through running simulations of load it was observed that the web-API could handle the inserts to the database, and manage all the concurrent requests coming from the simulated motes. Thus proving that the design and implementation can handle a large number of motes without a significant increase in cost given that the system was running on the $ 5 per month virtual machine from DigitalOcean. At this point we also saw that the performance of the view decreased as more data points where added. This could have been remedied by aggregating the data to a larger extent or rendering the graphs server side. The system still performed well when looking at graphs on a 24 hour view, so it was decided that for the purposes of proving that the data could be displayed in an intuitive manner for a single location this front end rendering of the data was sufficient.

8.4.3 Protocol testing

One of the original planned activities was to test the power longevity when using different protocols. This is the reason time was spent looking at the different options for IoT friendly protocols, when the scope of the work shifted away

from the performance of the motes over to an overall implementation with an increased focus on the design and considerations this testing was left out of the final project. Comparing the protocols listed in this thesis (HTTP, CoAP, MQTT, AMQP) will be left for future work. Given that the power constraint was considered out of scope, the resource cost of HTTP was no longer a consideration within the scope of the work.

8.5 Criteria overview

The thesis describes the design, development and deployment of a platform for gathering and visualizing data. Described as a ideal design and the implementation that followed. The initial requirements where that the platform should handle data from a large set of motes, that it should visualize the data and through the visualization provide value and insight. All this should be done at a low cost. Through the design it is described how the system could scale, and by implementing the data simulation it is showed that the system could handle a large amount of sensors providing data to the platform. In this chapter it is described how each of the requirements for the system where tested during different activities, and what was learned through said activities. Through this proving the solution has the characteristics required to meet the requirement set in the introduction chapter.

We have seen screenshots (fig 7.1 and 8.2) from the application that it is possible to visualize the soil moisture of a plants soil over time. And that the graph is easy to understand. Through this it is demonstrated that the system is able to visualize the data, and that it is possible to see clear trends in the data over time.

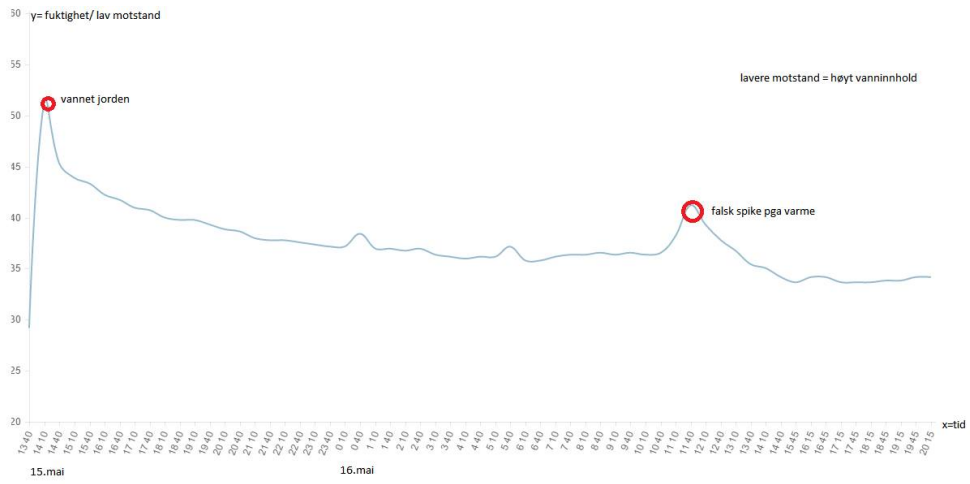
By meeting these technical and functional requirements the system that was designed and built could aid in establishing a baseline for excess irrigation and give a clear picture of both when a farmer would need to water, but also when it is not necessary due to rainfall or other weather conditions that keep the soil at a high moisture content. It achieves this through easily accessible information and visualization. However documenting this in a deployment setting would require work that is outside the scope of this thesis. It is unfortunate that the thesis was not able to find clear numbers of a true definition of the price of gathering this data in the past, and doing a comparison of the price of each data point and the

value of the information will go in to the suggested future work for this thesis.

8.6 Is it practically possible to implement a system for monitoring soil moisture

Through this thesis it has been shown that a proof of concept version of the system can be implemented by one to two people using commodity hardware, open source software packages and cheap cloud solutions. The thesis has outlined and explained that it can be deployed, and that one can clearly see the soil moisture levels. See fig 8.2 for hour by hour moisture example

Figure 8.2: Soil moisture hour by hour



8.7 Is sensor technology a reasonable way to address excessive irrigation

Through this thesis this has been one of the central question that was set out to answer. This thesis has looked at what excessive irrigation is, and some of the problems that can come along with excessive use of water. First excessive

irrigation was defined as a management problem and not a mechanical problem with the help of the article by Amy Lilien-feld and Mette Asmild [20]. With this knowledge we looked at the changes in ease in collecting important data that we can use in getting excessive irrigation under control.

My conclusion is that sensor technology undoubtedly will be a good tool in the arsenal of a farmer who is concerned about water use, both from a profit and environmental perspective. The thesis has shown that sensor networks can be implemented and used for collecting data that used to be hard and expensive to come by.

8.8 Can a system for monitoring water consumption be implemented with limited resource use

In the context of this thesis the focus is on resources defined as time and money. The work that was done in the proof of concept was to show that the goal of monitoring water consumption can be done by developers with limited professional experience and knowledge about sensors and electrical engineering. Although there is a lot of trial and error involved in developing any product or software, my conclusion is that this type of system can be implemented within a reasonable budget, and that building a full scale version of this system is something a small team would be able to do. Both from the ground up, or by leveraging some of the services described in this thesis.

Chapter 9

Closing reflections and future work

This chapter aims to add closing remarks to the implementation and thesis work. It also includes further work that is relevant for improving the system and increasing the impact of the system implementation.

9.1 Summary

In this thesis we have explored the concepts and technologies that make up Cloud Computing and Internet of Things, we have explored in depth the configuration of Wireless Sensor Networks, and examples of how these technologies could be applied in agricultural scenarios to monitor soil moisture. We see from these examples that applying the knowledge of agriculturalists is crucial to the success of the efforts and that the knowledge derived from the systems help the users make the right decisions. It could be relevant to further explore how to make these technologies more efficient, how they can become more accessible to users and how they can be applied to emerging agricultural techniques like Urban Farming, Hydroponics, Aquaponics and Vertical Farming. Primarily the area of interest is to implement this system in a Norwegian context as we have challenges emerging with long droughts during summer in recent years and very wet periods during fall. My intuition tells me that a lot can be done to increase the yield of livestock feed crops like grass, which has proven highly important in a Norwegian context.

In the end it became clear that the value of the data collected was limited, but in

so far as answering the question of whether or not this system is feasible it helped build the conclusion. The value came primarily from the visualization platform that was built as a part of the reference implementation of this system. This yielded a clear picture of the state of the monitored plants, and gave clear insight that could potentially help with watering in a large scale agricultural operation of similar characteristics.

- Is sensor technology a reasonable way to address excessive irrigation?
  - Based on improved information and visualization we can reduce excess irrigation through fact based decision making.
- Can a system for monitoring soil moisture be implemented with limited resource use?
  - Through exploring cost and difficulty of implementing a system for soil monitoring we can see that it is possible to implement with limited resource use.
- Is it practically possible to implement a system for monitoring soil moisture?
  - Based on the design and implementation meeting the requirements through the testing activities we can see that it is practically possible to implement a proof of concept system for soil moisture monitoring.

9.2 Future work

Something that became apparent throughout the project was that my initial assumptions regarding what factors to monitor where too limited to give an overall picture of the health of a major farming operation. Something as basic as what depth to put the sensors at can become a major study in itself and it would have benefited from research and discovery around this. Although the thesis did not go to deep in to the plant science of water monitoring, we have learned that there are many factors that contribute to the state of a plant. And we have also learned that there are many factors that impact water retention in soil. Is the soil sandy, is it hard packed, what level of fertilizer is present. Learning more

about the different factors, and monitoring how the different variables impact the reading would have been interesting.

It is also natural to think that we would want to do more with the platform that was developed. A few of the following items come to mind, security, automation of deployment and user and role engineering. These things where not applicable in a proof of concept scenario, but would have been a must have to launch this system in a production setting.

Mote software deployment is something that was immediately seen as a potential problem. The deployment model for software could have been a massive problem in a production setting. I had some ideas on how to improve this, but none that where possible to implement with the architecture chosen for the project. Here it would be interesting to invest more time in to the data transfer protocol and the network topology. As an example utilizing edge nodes with WiFi and running the rest of the nodes on radio signals could reduce the points in the system needed to change configuration and update software. As an example, problems with the HTTP protocol implementation or the system needs changes or updates to the code. The edge node could be targeted for update as all the connectivity in one node at the edge as a gateway to the server. The goal of this suggested further work would be to achieve a greater separation of concerns, having the motes focus on data gathering and allow the edge nodes acting as gateways to deal with connectivity, data caching, uploading data. As well as limiting the number of nodes in the network that needs to change during an update to the software or change the technology of the APIs at any point in the future.

It would also be important to research the price of collecting agricultural data in the past. This would help me to quantify the cost reduction that the proof of concept gives compared to existing methods. Quantifying this reduction could have helped me underpin the conclusions of the thesis even better, and would be an important step in relating the proof of concept to the real world implementation costs of existing solutions. After reaching out to Mette Asmild to ask if they had kept any data on the cost of gathering agricultural data through field studies, it became clear that this would have to be quantified in another way as they did not have as they unsurprisingly did not have any data on this almost 12 years after their paper was published.

There is a lot of room for statistical analysis on the data sets that the sensors generate. Based on aggregate level data for more variables the system should be able to start analyzing their relationship and see what factors have a high correlation in the model. Over time the hope would be to link this to weather data and build a model that could help in predicting and optimizing irrigation over time. This was an area it would have been of interest to research further, but due to time constraints it is ending up in future work. Building a machine learning model based on the data and using factors such as sun hours and weather forecasts it should be possible to say something about the rate of consumption and evaporation. Resulting in more accurate and precise irrigation use, thus contributing to reduced water usage and ensuring limitation of excessive irrigation.

Bibliography

[1] M. Aazam and E. N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *2014 International Conference on Future Internet of Things and Cloud*, Aug. 2014, pp. 464–470. DOI: 10.1109/FiCloud.2014.83.

[2] AMQP, *Amqp is the internet protocol for business messaging*, https://www.amqp.org/about/what, Jun. 2017.

[3] AWS, *10 years of amazon web services*, https://aws.amazon.com/10year/, Accessed: 31.10.2019.

[4] ——, *Aws regions*, https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html, Accessed: 09.11.2019.

[5] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen, *RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0*, http://www.faqs.org/rfcs/rfc1945.html, May 1996.

[6] C. Bormann, K. Hartke, and Z. Shelby, *The Constrained Application Protocol (CoAP)*, RFC 7252, Jun. 2014. DOI: 10.17487/rfc7252. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt.

[7] Bosch, *Iot based smart irrigation system*, https://www.bosch.com/stories/iot-based-smart-irrigation-system/, Accessed: 24.10.2019.

[8] J. Burrell, T. Brooke, and R. Beckwith, "Vineyard computing: Sensor networks in agricultural production," *IEEE Pervasive Computing*, vol. 3, no. 1, pp. 38–45, Jan. 2004, ISSN: 1536-1268. DOI: 10.1109/MPRV.2004.1269130.

[9] CDC, *Types of agricultural water use*, https://www.cdc.gov/healthywater/other/agricultural/types.html, Accessed: 24.10.2019.

[10] M. Delano, *Roundup ready crops*, https://web.mit.edu/demoscience/Monsanto/about.html, Accessed: 09.11.2019.

[11] ericsson, *Internet of things forecast*, https://www.ericsson.com/en/mobility-report/internet-of-things-forecast, Accessed: 10.11.2019.

[12] espressif, *Esp8266ex datasheet*, https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf, Accessed: 27.10.2019.

[13] Gartner, *Gartner says worldwide iaas public cloud services market grew 31.3% in 2018*, https://www.gartner.com/en/newsroom/press-releases/2019-07-29-gartner-says-worldwide-iaas-public-cloud-services-market-grew-31point3-percent-in-2018, Accessed: 27.10.2019.

[14] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013, Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services amp; Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2013.01.010". [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241.

[15] J. Gutiérrez, J. F. Villa-Medina, A. Nieto-Garibay, and M. Á. Porta-Gándara, "Automated irrigation system using a wireless sensor network and gprs module," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 1, pp. 166–176, Jan. 2014, ISSN: 0018-9456. DOI: 10.1109/TIM.2013.2276487.

[16] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11–17, 2015.

[17] T. Khokhar, *Chart: Globally, 70 % of freshwater is used for agriculture*, https://blogs.worldbank.org/opendata/chart-globally-70-freshwater-used-agriculture, Accessed: 27.10.2019.

[18] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the internet of things with coap," in *2014 International Conference on the Internet of Things (IOT)*, Oct. 2014, pp. 1–6. DOI: 10.1109/IOT.2014.7030106.

[19] K. Lee, D. Murray, D. Hughes, and W. Joosen, "Extending sensor networks into the cloud using amazon web services," in *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*, Nov. 2010, pp. 1–7. DOI: 10.1109/NESEA.2010.5678063.

[20] A. Lilienfeld and M. Asmild, "Estimation of excess water use in irrigated agriculture: A data envelopment analysis approach," *Agricultural Water Management*, vol. 94, no. 1, pp. 73–82, 2007, ISSN: 0378-3774. DOI: https://doi.org/10.1016/j.agwat.2007.08.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378377407002119.

[21] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of wireless sensor networks towards the internet of things: A survey," in *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, Sep. 2011, pp. 1–6.

[22] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," Gaithersburg, MD, United States, Tech. Rep., 2011.

[23] MotherBoard, *Tractor hacking: The farmers breaking big tech's repair monopoly*, https://www.youtube.com/watch?v=F8JCh0owT4w, Accessed: 26.10.2019.

[24] NBIM, *Water management, expectations to companies*, https://www.nbim.no/contentassets/228e2216a0c34c18a48df90c98a39a13/nbim-water-management-expectation-document.pdf, Accessed: 31.10.2019.

[25] S. Notra, M. Siddiqi, H. H. Gharakheili, V. Sivaraman, and R. Boreli, "An experimental study of security and privacy risks with emerging household appliances," in *2014 IEEE Conference on Communications and Network Security*, Oct. 2014, pp. 79–84. DOI: 10.1109/CNS.2014.6997469.

[26] nve, *Tørke og flomåret 2018*, https://www.nve.no/nytt-fra-nve/nyheter-hydrologi/torke-og-flomaret-2018/, Accessed: 26.10.2019.

[27] OECD, *Agriculture and water*, http://www.oecd.org/agriculture/ministerial/background/notes/5_background_note.pdf, Accessed: 31.10.2019.

[28] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges," *Computers and Electronics in Agriculture*, vol. 118, p. 68, 2015, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2015.08.011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169915002379.

[29] OneTick, *Onetick timeseries database*, https://www.onetick.com/time-series-database-tick-data-onetick, Accessed: 24.10.2019.

[30] F. Pierce and T. Elliott, "Regional and on-farm wireless sensor networks for agricultural systems in eastern washington," *Computers and Electronics in Agriculture*, vol. 61, no. 1, pp. 32–43, 2008, Emerging Technologies For Real-time and Integrated Agriculture Decisions, ISSN: 0168-1699. DOI: http://dx.doi.org/10.1016/j.compag.2007.05.007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169907001664.

[31] postman, *Postman homepage*, https://www.getpostman.com/, Accessed: 10.11.2019.

[32] Schneider, *Water resources*, https://www.schneider-electric.com/en/work/solutions/for-business/water/water-resources/, Accessed: 24.10.2019.

[33] sensorion, *Datasheet sht1x (sht10, sht11, sht15)*, https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf, Accessed: 27.10.2019.

[34] P. C. P. D. Silva and P. C. A. D. Silva, "Ipanera: An industry 4.0 based architecture for distributed soil-less food production systems," in *2016 Manufacturing Industrial Engineering Symposium (MIES)*, Oct. 2016, pp. 1–5. DOI: 10.1109/MIES.2016.7780266.

[35] tableu, *Tableu homepage*, https://www.tableau.com/, Accessed: 10.11.2019.

[36] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, May 2005, ISSN: 0018-9162. DOI: 10.1109/MC.2005.163.

[37] UN, *United nations development goals*, un.org/sustainabledevelopment/water-and-sanitation/, Accessed: 09.11.2019.

[38] veracity, *Veracity homepage*, https://www.veracity.com/, Accessed: 10.11.2019.

[39] Wildeye, *Wildeye homepage*, https://www.mywildeye.com/, Accessed: 03.11.2019.

[40] I. S. chapter "World fresh water resources" in Peter H. Gleick (editor), *Water in crisis*. New York: Oxford University Press, 1993.

[41] Y. Xing and Y. Zhan, "Virtualization and cloud computing," in *Future Wireless Networks and Information Systems*, Y. Zhang, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 305–312, ISBN: 978-3-642-27323-0.

[42] user: zdebel, *Www.esp8266.com forum post*, https://www.esp8266.com/viewtopic.php?f=19&t=4212, Accessed: 12.11.2019.

Appendix Code

init.lua

The init .lua file is the file resposible for connecting do the WiFi network and running the deep sleep function of the nodemcu.

```
-- wait for an IP
cnt = 10
tmr.alarm(0,500,1,function()
  if wifi.sta.getip()==nil then
    cnt = cnt-1
    if cnt<=0 then
      -- Did not get an IP in time, so quitting
      tmr.stop(0)
      gpio.write(pin_error,1)
      node.dsleep(1 * 60 * 1000000)
      --print "Not connected to wifi."
    end
  else

    -- Connected to the wifi
    tmr.stop(0)
    --print("\nStarting main")
    dofile("main.lua")
  end
end)
```

sth11.lua

This is an included module [42]

```
local moduleName = ""
local M = {}
_G[moduleName] = M

```

```
--table module
local table = table
--bit module
local bit = bit
--timer module
local tmr = tmr
--gpio module
local gpio = gpio
-- local environoment
setfenv(1, M)

local function set_sck(v)
    if (v == 1) then
        gpio.write(sck, gpio.HIGH)
    else
        gpio.write(sck, gpio.LOW)
    end
end

local function set_data(v)
    if (v == 1) then
        gpio.mode(data, gpio.INPUT)
    else
        gpio.write(data, gpio.LOW)
        gpio.mode(data, gpio.OUTPUT)
    end
end

local function start()
    set_data(1)
    set_sck(0)
    tmr.delay(1)
    set_sck(1)
    tmr.delay(1)
    set_data(0)
    tmr.delay(1)
    set_sck(0)
    tmr.delay(1)
    set_sck(1)
    tmr.delay(1)
    set_data(1)
    tmr.delay(1)
    set_sck(0)
end

local function reset()
    set_data(1)
    set_sck(0)
    for i = 0, 8 do
```

```
        set_sck(1)
        tmr.delay(1)
        set_sck(0)
        tmr.delay(1)
    end
    start()
end

local function write(v)
    local c = v
    local ret
    for i = 0, 7 do
        if (bit.band(c, 0x80) ~= 0) then
            set_data(1)
        else
            set_data(0)
        end
        c = bit.lshift(c, 1)
        set_sck(1)
        tmr.delay(1)
        set_sck(0)
        tmr.delay(1)
    end
    set_data(1)
    set_sck(1)
    tmr.delay(1)
    ret = gpio.read(data)
    set_sck(0)
    return ret
end

local function read(ack)
    local c = 0
    set_data(1)
    for i = 0, 7 do
        c = bit.lshift(c, 1)
        set_sck(1)
        tmr.delay(1)
        c = bit.bor(c, gpio.read(data))
        set_sck(0)
        tmr.delay(1)
    end
    if (ack == 1) then
        set_data(0)
    end
    set_sck(1)
    tmr.delay(1)
    set_sck(0)
    set_data(1)
```

```
    return c
end

function M.init(s, d)
    sck = s
    data = d
    gpio.mode(data, gpio.INPUT);
    gpio.mode(sck, gpio.OUTPUT);

end

function M.read_measurements()

    reset()
    start()
    --TODO: add ACK check
    write(0x03)

    set_data(1)

    --TODO: add timeout
    for i = 0, 65536 do
        if (gpio.read(data) == 0) then
            break
        end
    end

    t0 = read(1)
    t1 = read(1)
    --TODO: add crc checking
    t_crc = read(0)
    t = bit.lshift(t0, 8);
    t = bit.bor(t, t1)
    --t = t - 4010 --for 5V
    t = t - 3970 --for 3.5V

    start()
    write(0x05)
    set_data(1)

    --TODO: add timeout
    for i = 0, 65536 do
        if (gpio.read(data) == 0) then
            break
        end
    end

    h0 = read(1)
    h1 = read(1)
```

```
    --TODO: add crc checking
    h_crc = read(0)

    h = bit.lshift(h0, 8);
    h = bit.bor(h, h1)

    --compensation magic part 1
    rh = 268 * h
    rh = rh / 256
    rh = -rh
    rh = rh + 24052
    rh = rh * h
    rh = rh - 1341391
    rh = rh / 65536

    --compensation magic part 2
    rh_true = 52 * h
    rh_true = 6554 + rh_true
    rh_true = rh_true * (t - 2500) / 100
    rh_true = rh_true / 65536
    rh_true = rh_true + rh

    return t, rh_true
end

return M
```

main.lua

main.lua is the file refenced in init.lua and is responsible for reading and sending the moisture data from the sensor

```
function sendData()
    sht = require("sht11")
    --clock on GPIO4, data on GPIO12
    sht.init(3, 4)

    t, h = sht.read_measurements()
    print(wifi.sta.getip())

    chipId = node.chipid()
    soilMoisture = h/10
    soilTemperature = t/100

    --dataString = string.format('{"chipcode":"%f","soilMoisture":"%f", "
    soilTemperature":"%f"}', chipId, soilMoisture, soilTemperature)
```

```
    dataString = "{\"chipcode\": \"".. tostring(chipcode) .."\", \"soilMoisture\":
    \"" .. tostring(soilMoisture) .. "\", \"soilTemperature\": \"" .. tostring(
    soilTemperature) .. "\"}"
    --print(soilMoisture)
    --print(soilTemperature)

    http.post('http://192.81.221.165:80/api/datapoint',
          'Content-Type: application/json\r\n',
            dataString,
          function(code, data)
            if (code < 0) then
              --print("HTTP request failed")
              node.dsleep(1 * 60 * 1000000)--sleep 1 minute and retry
            else
              print(code, data)
              node.dsleep(30 * 60 * 1000000)--sleep 30 minutes
            end
          end)


end

sendData()
```

server.js

Server.js is responsible for recieving and saving the data sent from the motes.

```
// server.js

// BASE SETUP
// =============================================================================

// call the packages we need
var express    = require('express');        // call express
var app        = express();                 // define our app using express
var bodyParser = require('body-parser');
var cors = require('cors');// call express

// configure app to use bodyParser()
// this will let us get the data from a POST
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(cors());

var port = process.env.PORT || 8080;        // set our port

```

```
var moment = require('moment');
var Plant = require('./models/plant');
var Water = require('./models/water');

// ROUTES FOR OUR API
// =============================================================================
var router = express.Router();              // get an instance of the express Router


router.use(function(req, res, next) {
    // do logging
    console.log('Something is happening.');
    next(); // make sure we go to the next routes and don't stop here
});

router.route('/plant')

    // create a bear (accessed at POST http://localhost:8080/api/bears)
    .post(function(req, res) {

        var plant = new Plant();      // create a new instance of the Bear model

        plant.chipcode = req.body.chipcode;
        plant.soilMoisture = req.body.soilMoisture;
        plant.time = Date.now();

        console.log(plant)

        // save the bear and check for errors
        plant.save(function(err) {

            if (err)
                res.send(err);

            res.json({ message: 'datatapoint created!' });

        });

    }).get(function(req, res) {


    Plant.find(function(err, data){
        res.json(data)
    })


});

router.route('/editplant')
```

```
69
70      // create a bear (accessed at POST http://localhost:8080/api/bears)
71      .get(function(req, res) {
72
73          var plant = new Plant;
74
75          Plant.find(function(err, plant){
76
77              console.log(plant.length)
78
79              currenttime = moment();
80
81              timepassed = (30 * plant.length);
82
83              for(i = 0; i < plant.length; i++){
84
85                  plant[i].time = moment(currenttime).subtract(timepassed, 's')
86
87                  plant[i].save();
88
89                  timepassed = timepassed - 30;
90
91              }
92
93              console.log(plant)
94
95
96          })
97
98
99
100     });
101
102 router.route('/addwater')
103
104     // create a bear (accessed at POST http://localhost:8080/api/bears)
105     .post(function(req, res) {
106
107         var water = new Water();      // create a new instance of the Bear model
108
109         water.water = false;
110         water.time = Date.now();
111
112         console.log(water)
113
114         // save the bear and check for errors
115         water.save(function(err) {
116
117             if (err)
```

```
                res.send(err);

            res.json({ message: 'water created!' });

        });
});

router.route('/editwater')
    // create a bear (accessed at POST http://localhost:8080/api/bears)
    .post(function(req, res) {

        var water = new Water();

        Water.find(function (err, data) {

            water.water = req.body.water;
            water.time = Date.now();

            console.log(water);

            water.save();

        })

    });



// test route to make sure everything is working (accessed at GET http://localhost
    :8080/api)
router.get('/', function(req, res) {
    res.json({ message: 'hooray! welcome to our api!' });
});

// more routes for our API will happen here

// REGISTER OUR ROUTES -------------------------------
// all of our routes will be prefixed with /api
app.use('/api', router);

// START THE SERVER
// =============================================================================
app.listen(port);
console.log('Magic happens on port ' + port);
var mongoose   = require('mongoose');
mongoose.connect('mongodb://178.62.194.135/greenhouse'); // connect to our database
/*mongodb://ip/database*/
```

index.html

Index.html is the main file to control the front end

```
<!-- index.html -->
<!DOCTYPE html>
<html ng-app="App">
<head>
    <!-- SCROLLS -->

    <meta charset="UTF-8">

    <!-- load bootstrap and fontawesome via CDN -->
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.css" />

    <link href='https://fonts.googleapis.com/css?family=Lobster' rel='stylesheet' type='text/css'>

    <!-- SPELLS -->
    <!-- load angular and angular route via CDN -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0-beta.2/angular.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0-beta.2/angular-sanitize.js"></script>
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular-route.js"></script>
    <script src="https://cdn.rawgit.com/google/code-prettify/master/loader/run_prettify.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.0/angular-cookies.js"></script>
    <script type="text/javascript" src="https://cdn.rawgit.com/auth0/angular-jwt/master/dist/angular-jwt.js"></script>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/angular-resource/1.5.5/angular-resource.min.js"></script>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/1.0.2/Chart.js"></script>
    <script src="https://cdn.jsdelivr.net/angular.chartjs/0.8.8/angular-chart.js"></script>
    <!-- load ngmessages -->
    <script src='http://code.angularjs.org/1.3.0-beta.8/angular-messages.js'></script>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>

    <script src="../node_modules/angular-momentjs/angular-momentjs.js"></script>

```

```
    <link rel="stylesheet" href="../node_modules/office-ui-fabric/dist/css/fabric.min.css">
    <link rel="stylesheet" href="../node_modules/office-ui-fabric/dist/css/fabric.components.css">

    <script src="../node_modules/office-ui-fabric/dist/js/jquery.fabric.js"></script>
    <link rel="stylesheet" href="../frontend/css/custom.css">




    <script src="pageController.js"></script>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/angular.chartjs/0.8.8/angular-chart.css">

    <!-- Custom CSS -->

</head>


<body>

<div>
    <!-- HEADER AND NAVBAR --
        <!-- Sidebar -->

    <div class="ms-NavBar">
        <div class="ms-NavBar-openMenu js-openMenu"><i class="ms-Icon ms-Icon--menu"></i></div>
        <div class="ms-Overlay"></div>
            <li class="ms-NavBar-item"><a class="ms-NavBar-link">About the project</a></li>
            <li class="ms-NavBar-item"><a class="ms-NavBar-link">Soil status</a></li>
            <li class="ms-NavBar-item ms-NavBar-item--right"><a class="ms-NavBar-link"><i class="ms-Icon ms-Icon--mail"></i> Contact</a></li>
        </ul>
    </div>

    <!-- MAIN CONTENT AND INJECTED VIEWS -->
    <div id="page-content-wrapper">

        <!-- angular templating -->
        <!-- this is where content will be injected -->

        <div ng-view></div>

    </div>
```

```
</div>

</body>
</html>
```

dashboard.html

dashboard.html shows the tables and information for the water usage

```

<div class="ms-Grid spacer" ng-init="init()">
    <div class="ms-Grid-row">

        <div class="ms-Grid-col ms-u-md2 ms-u-mdOffset2">


            <div class="ms-PersonaCard">
                <div class="ms-PersonaCard-persona">
                    <div class="ms-Persona ms-Persona--xl">
                        <div class="ms-Persona-imageArea">
                            <div class="ms-Persona-initials ms-Persona-initials--blue">AL</div>
                            <img class="ms-Persona-image" src="../frontend/persona/erlend.jpg"></div>
                        <div class="ms-Persona-presence"></div>
                        <div class="ms-Persona-details">
                            <div class="ms-Persona-primaryText" title="Erlend Westbye">Erlend Westbye</div>
                            <div class="ms-Persona-secondaryText">Really talented developer</div>
                            <div class="ms-Persona-optionalText">Available 24/7</div>
                        </div>
                    </div>
                </div>


            </div>



        </div>


    <div class="ms-Grid-col ms-u-md6 ">

        <canvas id="line" class="chart chart-line" chart-data="data"
```

```
                chart-labels="labels"
                chart-legend="false" chart-series="series"
                chart-click="onClick" chart-options="options">
        </canvas>

    </div>

    </div>

</div>
```

pageController.js

pageController.js contains the controllor for managing the front end

```
var App = angular.module('App', ['ngRoute', 'ngMessages', 'ngSanitize', 'ngCookies',
    'angular-jwt','ngResource','chart.js','angular-momentjs']);

App.config(function ($routeProvider) {

    $routeProvider
    //admin
        .when('/', {templateUrl: 'pages/dashboard.html', controller: 'dashController
    '});

});

App.controller('dashController', function ($scope, $http) {

    $scope.init = function(){

        var req = {
            method: 'GET',
            url: 'http://40.69.89.94:8080/api/plant'
        };
        $http(req).then(function(out){

            console.log(out.data)

            $scope.a = [];
            $scope.b = [];

            $scope.x = [];
            $scope.y = [];



```

```
            for(i = 0; i < out.data.length; i++){

                //
                //if(i % 60 ==0){

                if(out.data[i].chipcode == 8969837.000000){

                    $scope.a.push((((68/out.data[i].soilMoisture)*100)));

                }

                if(out.data[i].chipcode == 1800741.000000){

                    $scope.x.push((((68/out.data[i].soilMoisture)*100)));

                }

                if(out.data[i].chipcode == 1796642.000000){

                    $scope.y.push((((68/out.data[i].soilMoisture)*100)));

                    //$scope.b.push(date.toLocaleString())

                    date = new Date(out.data[i].time)


                    if(i%10 == 0){$scope.b.push(date.toLocaleString())}
                    else {$scope.b.push('')}

                }




                    //$scope.b.push('KL: ' + date.getHours() + ' Dato: ' + date.
    getDate() + '/' + date.getUTCFullYear())

                //}



            }

            $scope.options = {

                datasetFill  : false,
                scaleShowGridLines: false,
                pointDot: false
            }
```

```
79
80              $scope.plants = $scope.a;
81
82              $scope.labels = $scope.b;
83
84              $scope.data = [
85                  $scope.a,$scope.x,$scope.y
86              ];
87              $scope.onClick = function (points, evt) {
88                  console.log(points, evt);
89              };
90
91          });
92
93
94
95
96      };
97  });
98
99
100 App.run(function ($rootScope) {
101
102     $rootScope.$on('$viewContentLoaded',function(){
103         jQuery('html, body').animate({ scrollTop: 0 }, 0);
104     });
105
106 });
```

mote simulator.py

Python mote simulator

```
1
2  import sched, time
3  import csv
4  import requests
5  import json
6  from random import randint
7
8  NUMBER_OF_AREAS = 2
9  NUMBER_OF_MOTES = 50
10 AREAS = []
11 DATA_SETS = []
12 DATA_FILES = ["data2_1.json","data2_1.json","data2_1.json"]
13 LIMITS = []
14 ITERATORS = []
```

```
#AREA_NAMES = ["Backyard 1", "Backyard 2", "Green House 1","Green House 2", "Green House 3", "Corn Field 1", "Cornfield 2", "Living Room", "Basement"]
#AREAS_NAMED = []

for i in range(len(DATA_FILES)):
  with open(DATA_FILES[i]) as data_file:
    data = json.load(data_file)
    DATA_SETS.append(data)
    LIMITS.append(len(data)-2)




for i in range(NUMBER_OF_AREAS):
  print(str(i))
  AREAS.append(randint(0, len(DATA_SETS)))
  ITERATORS.append(2)
  #AREAS_NAMED.append(AREA_NAMES[i])

print(AREAS)

s = sched.scheduler(time.time, time.sleep)



def emitt():
  for i in range(NUMBER_OF_AREAS):
    print("Area: " + str(i))
    for j in range(NUMBER_OF_MOTES):

      rand = randint(-1, 1)
      reading = DATA_SETS[AREAS[i]][ITERATORS[i]+rand]
      datastring = "{\"area\":\""+str(i)+"\", \"mote\":\""+str(j)+"\", \"soilMoisture\":\"" + reading.get("soilMoisture") + "\", \"soilTemperature\":\"" + reading.get("soilTemperature") + "\", \"numberOfAreas\":\""+str(NUMBER_OF_AREAS)+"\", \"numberOfMotes\":\""+ str(NUMBER_OF_MOTES)+"\"}"


      dataobject = json.loads(datastring)

      if(j== 10):
        #print("mote: " + str(j) +" Reads: "+ reading.get("soilMoisture"))
        print(dataobject)


      #print(reading)
      r = requests.post('http://192.81.221.165:81/api/datapoint2', auth=('user', 'pass'), data=dataobject)
```

```
59
60      ITERATORS[i] = ITERATORS[i] + 1
61
62      if (LIMITS[AREAS[i]] == ITERATORS[i]):
63        ITERATORS[i] = 2
64    s.enter(10, 1, emitt,())
65
66  #emitt()
67  s.enter(1, 1, emitt,())
68  s.run()
```

mote simulator.py v2

Python mote simulator

```
1   import sched, time
2   import csv
3   import requests
4   import json
5
6
7   print('Posting every 30min to http://192.81.221.165:8085/api/plant')
8   with open('data.json') as data_file:
9       data = json.load(data_file)
10
11  i = 0
12  s = sched.scheduler(time.time, time.sleep)
13
14  def do_something(sc,iterator):
15
16    print("Posting")
17      r = requests.post('http://192.81.221.165:8085/api/plant', auth=('user', 'pass'),
          data=data[iterator])
18      iterator = iterator + 1
19      if(i==701):
20        i = 0
21      r = requests.post('http://192.81.221.165:8085/api/plant', auth=('user', 'pass'),
          data=data[iterator])
22      iterator = iterator + 1
23      if(i==701):
24        i = 0
25      r = requests.post('http://192.81.221.165:8085/api/plant', auth=('user', 'pass'),
          data=data[iterator])
26      iterator = iterator + 1
27      if(i==701):
28        i = 0
29
```

```
30      s.enter(1800, 1, do_something, (sc,iterator))
31
32
33  s.enter(1800, 1, do_something, (s,i))
34  s.run()
```

SYN, SYN-ACK, ACK

Christoffer Gard Osen was my partner on this project for more than two years, but decided to leave UiO at the end of 2018. Without him I would have felt very lost through this project and would like to extend a big thanks and acknowledgment for all the good conversations, discussion and collaboration.