Internet of Things (IoT)

Nathan Glover

May 30, 2017

Contents

List of Figures

Introduction

The following report outlines the process taken while designing an implementing a IoT system for Wireless Data Networks at Curtin University.

1.1 Project Selection

The project I chose for this report was *"Monitoring environmental parameters for agriculture"*.

I made this choice as it is an area that sparks a lot of interest with me in both a professional and personal sense. I also think it's an interesting industry to target, as the customer market are typically not the kinds of people to jump on new technology the moment it hits the market. It will be an excellent challenge to develop a platform that will provide clear benefits and hopefully assist in taking an industry into the 21st century.

1.2 Project Goal

Develop a cloud based web platform that can be used to monitor and display trending data taken sensors suitable for agriculture. Some examples of sensor data for the base platform will be:

- Temperature
- Humidity
- Moisture

Data from these sensors are transmitted over medium/long range to a central controller who handles directing the raw data up to the cloud platform.

Business Model

Utilizing the helpful tool on [2]*canvanizer.com* I set out to define where my platform would appropriately fit into the current digital marketplace.

2.1 Business Model Proposal

2.1.1 Key Partners

Who are your key partners? Who are your key suppliers?

- **Farmers** - The proposed system will be used on the farmland owned and operated by the farmers. This makes them our primary customer
- **Sensor Manufacturers'** - The manufacturing pipeline with most likely be made up of a number of different hardware vendors. My platform will require the mass production of the sensors used in the farmland on the custom sites.

2.1.2 Key Activities

What are your key activities?

- **Cloud Platform Development** - Provide programming and database management to the customer facing cloud based web platform to assist in viewing the personalized sensor data.

2.1.3 Key Resources

What are your key resources?

- **Developers** - Web Frontend and System backend.

- **Database Engineers** - Working with the database foundations and liaising with developers to implement pipelines for data normalization

2.1.4 Value Proposition

What are your value propositions?

- **Historical Sensor Data** - Gives the customers a way to view their sensor data historically or in real-time.
- **Future Forecasting** - Gives the customer a ballpark estimate on what kinds of trends should be expected in the near or distant future. This will take in weather forecast details or perform machine learning on historical data to find and display data trends.

2.1.5 Customer Relationships

Your customer relationships?

- **Self-service Cloud Platform (Basic)** - Customer can login to their sites platform and view/manipulate data without my companies involvement.
- **Self-service Cloud Platform (Support)** - Customers can contact my companies help-desk to get assistance if there are problems, or questions in regards to displaying data.
- **Self-service Cloud Platform (Advanced)** - Customers may call or contact to get our engineers to develop new features for them. This comes with additional costs and feature feasibility assessments need to be performed before making promises to the customer.

2.1.6 Channels

Channels

- **Source Clients through Retail** - Speak with leading supermarkets about where they source their produce from. Can propose that if the supermarkets assist us in finding farming customers, the benefits of farmers better understanding their agriculture will trickle down as cost savings for the stores.
- **Finders Discount** - Existing customers can get a percentage off their monthly/yearly subscription if they recommend a new customer.

2.1.7 Customer Segments

Customer Segments

- **Farmers** - Farmers and Agriculture workers will be the primary targeted customer group. Large farming areas where large sensor arrays would really benefit the customer if they could view the data in real-time.
- **Industrial & Manufacturing** - Temperature, Sound, Vibration, etc... sensors could be used in plants to give real-time and historical feedback on the condition of different areas in the plant.
- **Home Consumers** - A long term goal could be to develop kits for home users to setup in their own gardens. Wouldn't be the primary source of revenue and likely would only be explored to spread the companies name globally.

2.1.8 Cost Structure

What about your cost structure?

- **Cloud Hosted Data Analytics Platform** - The sensors talk back with a controller who handles sending the raw data back to our cloud platform. This platform does all the data analytics and allows the customer to login and view their data in interesting and helpful ways. The platform has a monthly/yearly subscription fee attached to it on top of the installation cost for the sensor network on the customers sites. Usually the two are bundled together during the sales pitch to give the customer a lump cost per annum of running the system over a variety of hectares.

2.1.9 Revenue Streams

What are your revenue streams?

- **Cloud Analytics Platform** - The cloud hosted analytics platform is the avenue in which the customer will be billed for. The platform does complex analytics and data visualization; whilst also offering ways to export, aggregate and manipulate incoming streams of data from the sensor network on the farm site(s).
- **Cloud Platform Additional Features** - If a customer requires a new feature to be developed to allow them to view data in different ways, additional R&D costs may be pitched before work begins

to implement this requested feature. Depending on how specific the feature is to that customer site will determine whether the customer will be out of pocket more money. This is because the time spent developing would not be helpful to other customers on the platform.

- **Hardware Profit Margin** - Hardware sensors can be sold with a small margin. The revenue generated from these will be small but can start to add up when bulk orders are placed.
- **Hardware Setup & Site Maintenance** - General system support and maintenance will be bundled with the customer yearly support contract, however when big new site implementations are required additional costs will be charged to compensate national/international travel to the customers site and for hours spend installing systems.

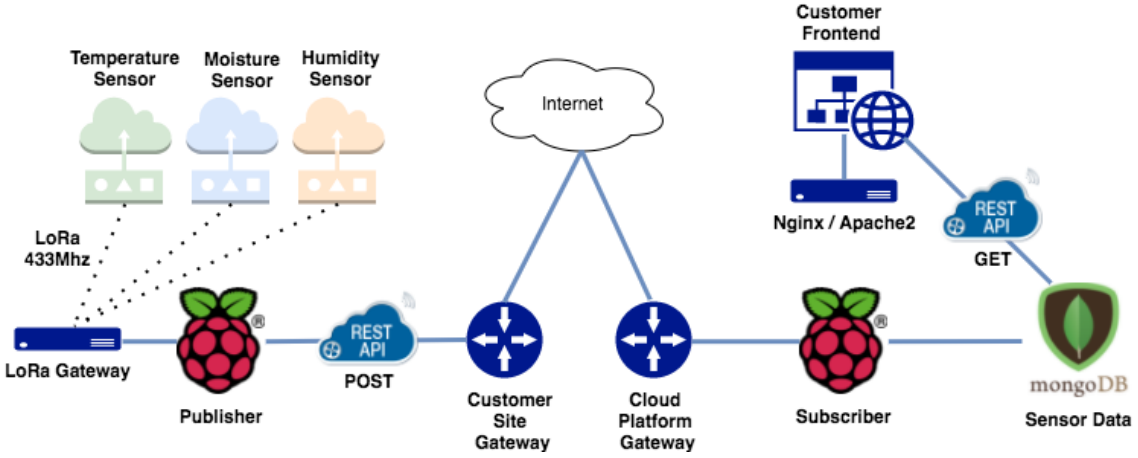
Design Proposal

In the following section I will outline the proposed design and attempt to justify the choices for my IoT system.

3.1 System Architecture

The system architecture specified below is a high level overview of how this system will be implemented. Technical details regarding the architecture can be found in the Design Architecture section later in the report.

Figure 3.1: Proposed system architecture



3.2 Design Justification

3.2.1 Sensing nodes

For my sensing nodes I have opted to use an Arduino Nano as my micro controller that will utilize the LoRaWAN communication protocols allowing an individual field sensor capabilities of sending up to 5km on the 433MHz spectrum.

A full parts list for proposed sensor can be seen below:

Item	Part No.	Cost p/u	Description
Nano CH340/ATmega328P	NANO-CH340G/ATmega328P	$2.49	Nano V3 constructed on the ATmega328 microcontroller (16MHz) has the small size and can be used together with model boards for solderless wiring (breadboard).
1200mAh Lithium Polymer	EHAO Li-polymer	$2.97	1200mAh Lithium cell battery at 3.7V
LoRa 20dBm transceiver	Lora610AES	$11.57	20dBm Radio transceiver LoRa Module, long range 4 5Km data transmission module with AES encryption.
Soil Moisture Sensor	Sens-SoilMoisture	$0.59	Soil moisture sensor for measuring the volumetric content of water in the soil.
MicroUSB Li-Ion Charger	TP4056-Li-Ion-charger	$0.39	The Micro-USB Li-Ion Battery Charger is a complete constant current/constant-voltage linear charger for single cell lithium-ion/Lithium Polymer (LiIon/LiPo) batteries.
Temp and Humidity sensor	DHT11	$1.54	This is DHT11 sensor for measuring temperature and humidity. It is accurate and inexpensive sensor that includes an ADC to convert analog values of humidity and temperature.

The design also integrates a 1200mAh Lithium cell battery that can be charged using the included

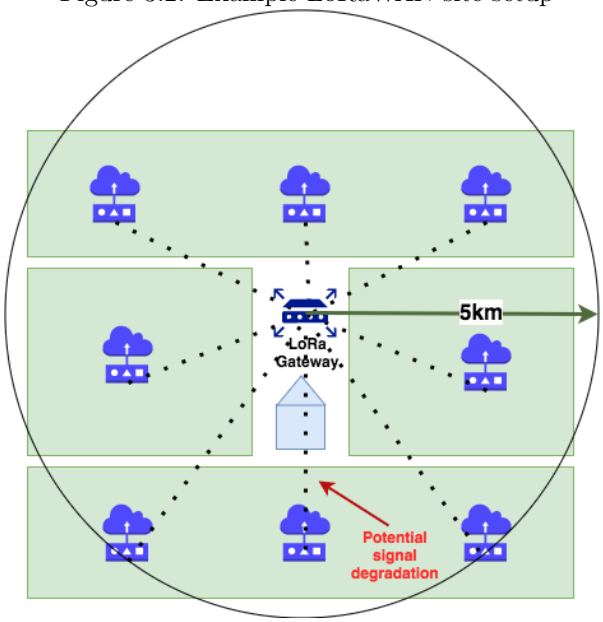
MicroUSB Li-Ion charger. Utilizing the low power mode on the Arduino chip can result in an expected period between charge of roughly 4-6months[3] depending on how often the sensor is polling data.

3.2.2 Placement of sensors

Due to the specifications of our chosen transmission medium (LoRa) we aren't limited by any difficult road-blocks when it comes to setting up our sensors. The frequency band of 433MHz has high penetration and does not normally require line of sight.

Take a look at the figure below for an example of a typical setup for a farm using LoRaWAN communications.

Figure 3.2: Example LoRaWAN site setup



Key points to note is that while all sensors fall within the 5km radius of the LoRa gateway, one of the sensors does not have direct line and is blocked by a tin cattle shed. Under most circumstances there should not be any issues with that sensor connecting back to the gateway, however it should be tested, and if communication is being dropped regularly then additional LoRa gateways can be installed.

***NOTE**: LoRa gateways as designed as packet forwarders. The packets received by the LoRa gateways are simply forwarded onto the LoRa network server which handles de-duplication of raw data packets. This*

means that having multiple gateways does not result in duplication of data.

3.2.3 Embedded Processor

As mentioned before, the embedded processor that has been selected for this project is the Arduino Nano. This processor was selected due to its wide use in the industry and support with a large number of existing open source libraries.

The main drawback of the Nano is that is comes packaged with a number of unused pins, and the MicroUSB controlled serves no real purpose once the controller is programmed. A way around this is to program the ATMEGA328 IC on an existing Arduino board, then pull the IC out and solder it only to the I/O pins we need for our implementation. This will reduce the space footprint of the device and also slightly lower the voltage drain from un-used components.

3.2.4 Energy Efficiency

The Arduino supports a low-power mode by lowering its base clock speed and utilizing an extended system sleep. This system sleep takes advantage of the hardwares Watch dog timer, allowing it to put itself to sleep for a given time.

Figure 3.3: Arduino Low-power mode[1]

Vcc (**V**)	Clock Speed (**MHz**)	Wake Current (**mA**)	Sleep Current (**uA**)
5.0	16	13.92	6.2
5.0	8	9.03	6.2
3.3	16	6.48	4.3
3.3	8	3.87	4.3

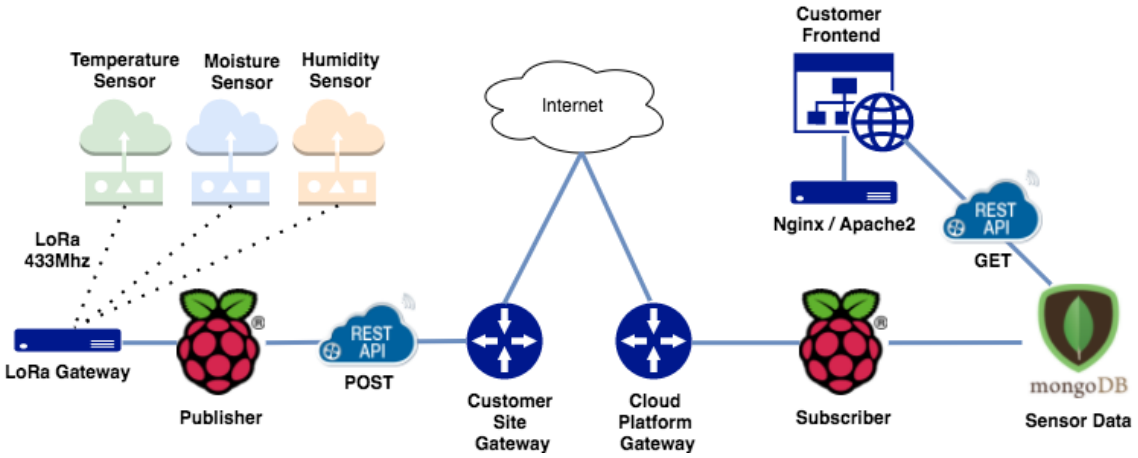
Using the figure above we can see that running the Arduino in sleep mode as often as possible is helpful to reduce the power usage. If we plan on taking a sensor reading once every minute and put the MCU into sleep mode in between this polling time, we can expect power saving resulting in the battery lasting roughly 7 times longer.

***NOTE**: the information above is based on running the system with just the ATMEGA328 IC and does not reflect the results of it when attached to the stock Arduino Nano* board.

3.2.5 Design Architecture

The design architecture displayed is the proposed landscape the platform will work with.

Figure 3.4: Proposed system architecture



1. Raw data is taken from the various sensor types and transmitted over the air using the LoRa protocol stack.

2. The data is received by the LoRa Gateway which forwards the traffic to the Raspberry Pi on-site that plays the role of LoRa Network Server.

3. The LoRa network server performs de-duplication on data packets that were picked up by adjacent gateways, then performs a RESTful POST outbound over the Internet to the subscriber node (also a Raspberry Pi) on our service platform.

4. The subscriber node decrypts the LoRa packet and formats the raw HEX payload into intelligible data. It uses a unique field (id) within the payload to determine which customer this packet came from.

5. The data is added as an entry to a customer specific MongoDB database as a document within a collection of similar data types.

6. A web-server running on Nginx serves up a number of scalable able instances of our front-end platform, using applications like Gunicorn to allow concurrent access to resources with threaded sessions.

7. When the web server goes is prompted display a particular set of data, It performs a RESTful GET on the MongoDB instance to query the customers data sets to display.

3.2.6 Scalability

There are three potential bottlenecks in my proposal that I would like to outline and justify:

LoRa Gateway

The LoRa enabled sensor can, when setup correctly, send a packet between the size of 2 to 255 octets[4] (8 bits per octet * 255 = 2050 bits or 255 bytes). The gateways themselves come in a variety of different configurations and over scaling levels of throughput. It is worth noting however, that a significant number of sensors all sending at the same time would be required in order to fill the gateways receive buffer. The main differentiation between gateways are their signal strength and form-factor.

MongoDB Query Latency

MongoDB is an open source database that classifies itself as NoSQL. It uses JSON-like documents with schema's to organize and collate data, with an emphasis on delivering query results extremely quickly. The requirements for our database would be high throughput of small items, which is exactly where MongoDB shines. The fast query times will also benefit customers with larger datasets that require extended historical data.

Frontend Availability

The frontend platform will be used by a number of customers all logging into their individual portal. My proposal is to have the homepage be static, providing only a method for logging in with credentials, and upon login a new instance of the platform is brought up using one of two methods:

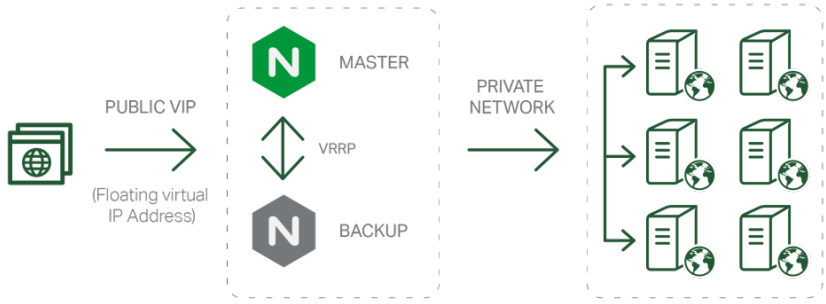
- **Docker** container, or any platform support by **Kubernetes**[5]. Instantiate instances of the web platform on an as needed basis. This method would require a lot of upfront configuration but would result in a more automated solution.
- **Gunicorn** with **Nginx** to scale up and down resources to the web platform. This method would require a little bit more manual management, but a lot easier to get up and running.

3.2.7 Reliability

When talking around reliability, a couple links can be made to the proposals made in order to make my platform scalable. For starters, the front-end availability method of instantiating new instances of the web

server can be expanded to utilize clustering across hosts. **Nginx**[6] for example supports a high availability architecture model that can be used to handle client session fail-over in the event a remote site goes down.

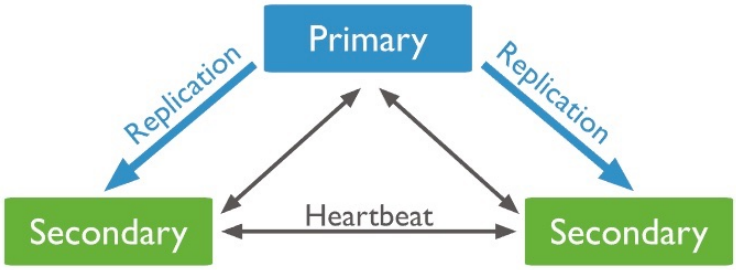
Figure 3.5: Nginx Clustering architecture



When we examine how the LoRa gateway design can be more reliable it becomes obvious that the best method of handling this is to throw more gateways into an area to create a dense mesh of gateways to handle varying loads of traffic. Luckily since de-duplication is handled by the LoRa network server, we don't need to worry about duplicate data points being captured as we scale up redundancy.

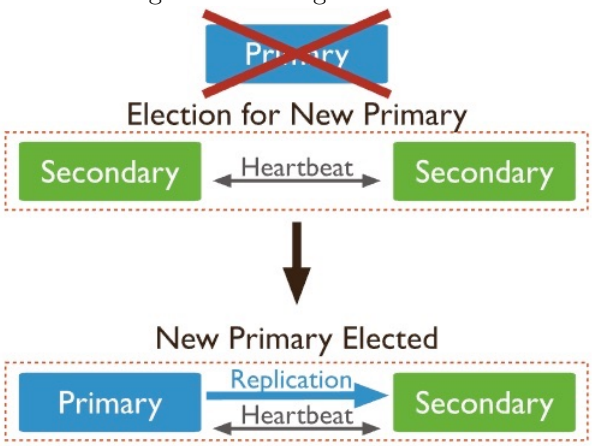
Reliability for **MongoDB** can be handled using its replication features. Entries are added to the primary endpoint and trickled down to an number of fail-over database instances.

Figure 3.6: MongoDB replication



In the event that the primary database goes offline, one of the secondary backup database instances is selected as the new candidate is promoted to master. The other secondary databases inherit this candidate as their new master and begin replicating of it.

Figure 3.7: MongoDB failover



This three solutions combined allow for a much more robust pipeline for our platform and also results in a more agile and modular solution going forward.

3.2.8 Cost Effectiveness

The pricing model for my platform depends on a few different components. To start with I looked at the costs of hosting on **AWS** as my web provider. Specifically I focused on the costs of low-tier hosting, as I can orchestrate new instances to start up and shutdown when needed.

Figure 3.8: AWS Low-tier pricing

	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t2.nano	1	Variable	0.5	EBS Only	$0.0075 per Hour
t2.micro	1	Variable	1	EBS Only	$0.015 per Hour
t2.small	1	Variable	2	EBS Only	$0.03 per Hour
t2.medium	2	Variable	4	EBS Only	$0.06 per Hour

Each customer could be expected to need between a **t2.small** to a **t2.medium**[7] depending on number of users visiting the front-end page and also the number of sensors and the frequency of data being inserted into the database. We could be safe and allocate $22.32 - $44.64 monthly towards customer platform hosting for a small farm with 10-15 sensors.

The hardware side would depend on what sensor was being used, but the flat cost for the embedded controller with the LoRa communication package comes to roughly $18.10 per device upfront cost. I would likely round this cost up to add some margin to the hardware for our time and R&D. The gateways cost depend on the range of the LoRa network, however a estimate for each gateway would be around $45.50.

The network server can technically be hosted in the cloud, however depending on the number of sensors on a site, it might be better to do the de-duplication of packets before sending it up to a cloud platform.

3.2.9 Network Architecture

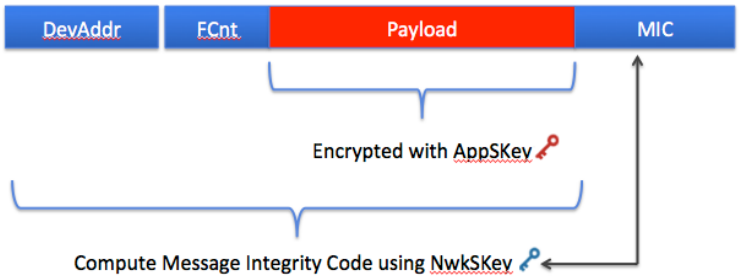
The network architecture isn't anything too complex and simply must include a way to route customer site data up to the Internet. My solution will easily tack onto existing consumer and enterprise grade network infrastructure and only requires a LAN connection from the LoRa gateways to the network directing Internet traffic.

A benefit of having the LoRa network server on-site is that it caches existing data packets from the sensors in the event that an Internet route goes down. All time-stamp information is encoded in the LoRa packet, so when the link comes back up there is not noticeable difference in the data collected during the outage.

3.2.10 Security

Security is where my solution, and the LoRa communication protocol really shines. A LoRa packet is encrypted in two ways using end-to-end AES-128[8].

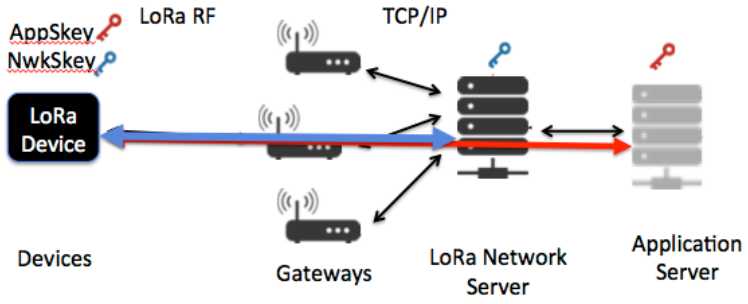
Figure 3.9: LoRaWAN Keysharing



When the LoRa enabled sensor first speaks with the LoRa Network server through a LoRa Gateway, it

is given a shared key that is used to encrypt the entire packet as it is sent from the sensor to the network server. This key is used to decode the full LoRa packet after it comes off the publicly readable RF spectrum.

Figure 3.10: LoRaWAN Decoding



The payload is extracted by the network server and send to the application server (in our case the subscriber node). The payload is encrypted using a pre-shared secret key that is used to extract the data from the encrypted payload. This results in a modular system of routing information that doesn't leave any room for accidental vulnerabilities, as the data is never in plain-text.

Prototype

In this part of the report I will discuss my implementation of something similar to the one outlined in the design proposal.

4.1 Hardware Used

- **Arduino Uno** - I used this micro-controller due to its availability and ease of use while prototyping.
- **Arduino Ethernet shield** - I used the ethernet shield in place of the LoRa communication stack while I prototype as it was a board I already had available and it is also easy to prototype on.
- **DHT11 sensor** - This sensor was cheap and provides temperature and humidity information. It is not overly expensive however so it will only do readings to the nearest degree Celsius which is not optimal for the final solution.
- **HP N54L MicroServer** - I used this to host my virtual machines for the data processing and web platform.

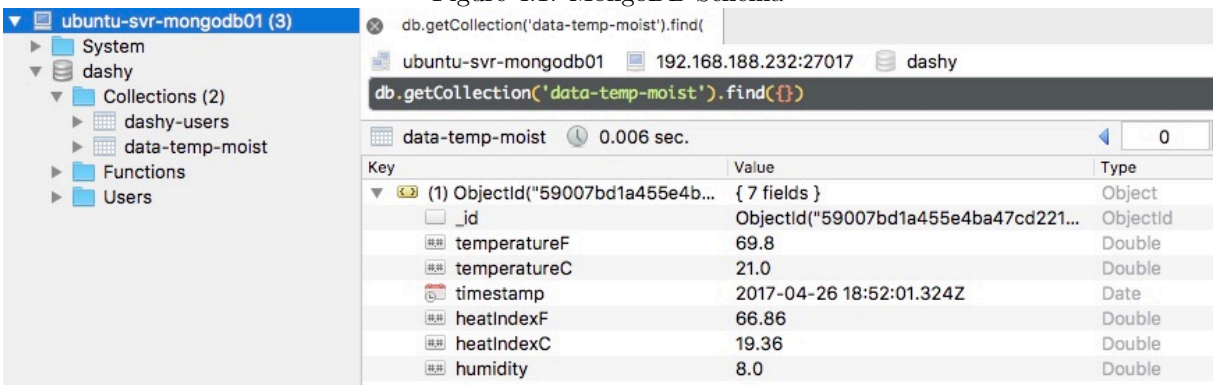
4.2 Software Used

- **Ubuntu 16.04** - Base operating system for my VMs.
- **Arduino editor** - Used to write and compile the Arduino code.
- **MongoDB** - Database software running to store sensor data.
- **Flask** - Python bases web framework for the front-end web page.
- **Gunicorn + Nginx** - Handles the web-page serving and session threading.

4.3 Database Schema

The schema used for my database is very simple, It is just a database called **dashy** (this is the temporary name I've given my platform) with two independent collections called **dashy-users** (stores salted SHA logins for the platform) and **data-temp-moist** (houses the temperature and humidity data).

Figure 4.1: MongoDB Schema



This database instance was setup and available only to my local network, as I did not need it to be public facing. The local IP address for it was **192.168.188.232** with a access port setup on **27017**.

4.4 Arduino Sensor reading and publishing

The Arduino was setup using the DHT and Ethernet libraries baked into the main Arduino libs. 90% of the code is main dealing with IP address association and serial data read/write functions for web service calls.

An important code block is the one that handles the reading of data from the DHT11 sensor.

```
// Reading temperature or humidity takes about 250 milliseconds!
// Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
humidity = dht.readHumidity();
// Read temperature as Celsius (the default)
temperatureC = dht.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
temperatureF = dht.readTemperature(true);

// Check if any reads failed and exit early (to try again).
if (isnan(humidity) || isnan(temperatureC) || isnan(temperatureF)) {
```

```
  Serial.println("Failed to read from DHT sensor!");
  return;
}

// Compute heat index in Fahrenheit (the default)
heatIndexF = dht.computeHeatIndex(temperatureF, humidity);
// Compute heat index in Celsius (isFahreheit = false)
heatIndexC = dht.computeHeatIndex(temperatureC, humidity, false);

```

Listing 4.1: Arduino code to pull data from the DHT11 sensor

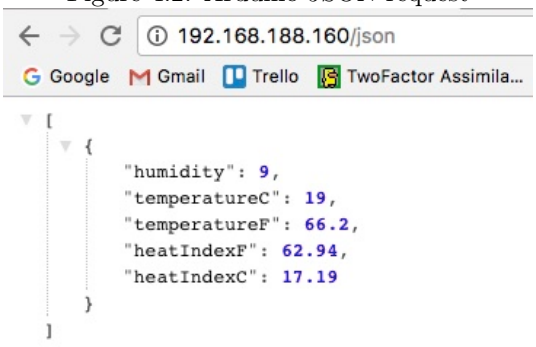
I decided on having the data be published over JSON in my prototype instead of making POSTs as I wanted the arduino system to be modular and it allowed me to use the data in other projects. The JSON information was encoded and sent to a client using the following code:

```
} else if (danweb_request("json")) {
  danweb_send_special_header(client, "application/json");

  String humidityStringJson = "{ \"humidity\":" + String(humidity) + ",";
  String temperatureCStringJson = "\"temperatureC\":" + String(temperatureC) + ",";
  String temperatureFStringJson = "\"temperatureF\":" + String(temperatureF) + ",";
  String heatIndexFStringJson = "\"heatIndexF\":" + String(heatIndexF) + ",";
  String heatIndexCStringJson = "\"heatIndexC\":" + String(heatIndexC) + "}";

  client.print("[");
  client.print(humidityStringJson);
  client.print(temperatureCStringJson);
  client.print(temperatureFStringJson);
  client.print(heatIndexFStringJson);
  client.print(heatIndexCStringJson);
  client.print("]");
```

Listing 4.2: Arduino JSON data encoding

This data is piped to any device that makes a RESTful GET request to the IP address of the Arduino (http://192.168.188.160/json). The output of this request can also be displayed in the browser by using the same address.

Figure 4.2: Arduino JSON request



4.5 Flask Platform Setup

I decided to run my platform on Flask as It was the web framework I was most knowledgeable with. I used the popular **Start Bootstrap - SB Admin 2** template with the **DataTables** jquery libraries to display information in table format.

Figure 4.3: Dashy database information display

Living Room Temperature & Humidity

Copy | Excel | CSV | PDF | Print

Search:

Timestamp	Temperature (°C)	Humidity (%)	Heat Index (°C)
Wed, 26 Apr 2017 23:59:04 GMT	20	9	18.29
Wed, 26 Apr 2017 23:58:05 GMT	20	9	18.29
Wed, 26 Apr 2017 23:57:02 GMT	20	9	18.29
Wed, 26 Apr 2017 23:56:03 GMT	20	9	18.29

4.5.1 MongoDB Population

In order to populate my MongoDB instance I also wrote a python script that runs as a scheduled task every minute to add an entry to the MongoDB database for my platform.

```
1 from pymongo import MongoClient
2 from datetime import datetime
```

```
import urllib, json

client = MongoClient('192.168.188.232:27017')
db = client['dashy']
col = db.get_collection('data-temp-moist')

response = urllib.urlopen("http://192.168.188.160/json")
data = json.loads(response.read())

post = {
  "humidity" : data[0]['humidity'],
  "temperatureC" : data[0]['temperatureC'],
  "temperatureF" : data[0]['temperatureF'],
  "heatIndexF" : data[0]['heatIndexF'],
  "heatIndexC" : data[0]['heatIndexC'],
  "timestamp" : datetime.now()
}

col.insert_one(post)
```

Listing 4.3: Arduino JSON data encoding

4.5.2 Network and DNS configuration

I set the instance up to run on **192.168.188.230** on my local network and port forwarded inbound **port 80** to this IP address as well. I registered a new A record for my personal domain in order to allow public resolution of my web platform outside of my home network.

Figure 4.4: DNS configuration

dashy.nathanglover.com	20	A	203.59.158.23	☐
www.nathanglover.com	20	CNAME	nathanglover.com.	☐

4.5.3 systemd service daemon management

To ensure my platform would run as a system daemon I added the following file **/etc/systemd/system/-dashy.service** and appended the following in order to ensure my platform uses the virtualenv I setup for it and runs with 3 workers for concurrent connection requests.

```
[Unit]
Description=Gunicorn instance to serve Dashy
After=network.target

[Service]
User=nathan
Group=www-data
WorkingDirectory=/home/nathan/Production/dashy
Environment="PATH=/home/nathan/.virtualenvs/flask-python2.7-dev/bin"
ExecStart=/home/nathan/.virtualenvs/flask-python2.7-dev/bin/gunicorn --workers 3 --bind unix
    :dashy.sock -m 007 wsgi:app

[Install]
WantedBy=multi-user.target
```

Listing 4.4: systemd manager script

Using the registered service I could also setup a shell script to automatically pull down changes to my front-end design and restart all the services required to push changes into production. I did this in the script below titled **dashy.sh**

```
#!/bin/bash
## Stop Systemctl service
sudo systemctl stop dashy

## Git Pull
git pull

## Restart Systemctl service
sudo systemctl start dashy
```

Listing 4.5: Dashy platform automated update production pipeline

4.5.4 Conclusion

Overall the prototype phase was a good proof of concept for how the front end and parts of the backend system will work. The idea I wasn't able to test was the LoRa communication protocol and network due to issues with receiving the parts within a reasonable time frame.

Something I would also like to change is the Arduino micro controller to instead to a NodeMcu[9] micro controller. This decision was primarily because of the lower cost and its on-board memory and file system that allows for the use of the Lua scripting language. Overall it is more technical to work with but offers many more options at a lower coat/size footprint.

I believe that the design, while simple in nature is a good foundation to move ahead with. It includes a reasonable cost analysis and the prototype shows that the platform is easy to setup and get running.

Bibliography

[1] A. T. GIANT. Reducing arduino power consumption. https://learn.sparkfun.com/tutorials/reducing-arduino-power-consumption.

[2] Canvanizer. Canvanizer - brainstorm better concepts. together with your team. https://canvanizer.com.

[3] A. Community. How to let your arduino go to sleep and wake up on an external event. http://playground.arduino.cc/Learning/ArduinoSleepCode.

[4] T. Clausen. A study of lora long range and low power networks for the internet of things. http://www.mdpi.com/1424-8220/16/9/1466/htm.

[5] Kubernetes. Production-grade container scheduling and management. https://github.com/kubernetes/kubernetes.

[6] Nginx. High availability for nginx plus. https://www.nginx.com/products/high-availability/.

[7] A. AWS. Amazon ec2 pricing. https://aws.amazon.com/ec2/pricing/on-demand/.

[8] T. Rosati. Lorawan security overview. http://www.trustpointinnovation.com/blog/2017/01/17/lorawan-security-overview/.

[9] NodeMcu. Nodemcu. http://nodemcu.com/index_en.html.