# *Remote IoT Weather Sensor Station*

Colton Powell
*Computer Engineering*
*Santa Clara University*
Santa Clara, California, U.S.
ctpowell@scu.edu

Justin Visas
*Engineering Management & Leadership*
*Santa Clara University*
Santa Clara, California, U.S.
jvisas@scu.edu

Matthew Salmanpour
*Electrical Engineering*
*Santa Clara University*
Santa Clara, California, U.S.
msalmanpour@scu.edu

Yezhou Zhao
*Computer Engineering*
*Santa Clara University*
Santa Clara, California, U.S.
yzhao.patrick@gmail.com

*Abstract*—**Smart homes, agriculture, and horticulture applications greatly benefit from IoT services to intelligently automate things such as turning blinds to control temperature, or controlling sprinklers to run during certain temperature and humidity levels. However, the sensor data needed to satisfy all of these applications is not always readily available. The Remote IoT Weather Sensor Station provides a solution to this problem. This project streams weather sensor data from multiple IoT devices to a remote MQTT broker, which forwards the data to web clients where the data is visualized. Additionally, the Remote IoT Weather Station is scalable, where nodes may be added or removed dynamically.**

## I. INTRODUCTION

IoT devices are well on the way to ubiquity and are rapidly expanding into markets and applications in which connectivity and intelligent automation have never been seen before. There are a number of factors driving this change: From an increased quality of life due to new capabilities enabled by this technology, to major corporations improving productivity and cutting costs. The value of IoT is found in its profound applicability in nearly all areas of life and industry.

The Remote IoT Weather Sensor Station focuses on creating a framework used primarily for promoting environmental awareness. Areas that serve to benefit most from this project are smart home, agricultural, and horticultural applications, as IoT devices can be used to intelligently automate functions such as turning on/off sprinklers, enabling HVAC, changing blind positions in a home, and more. These applications are not only extremely useful in terms of convenience, but also in terms of financial savings and environmental awareness.[1]

One glaring problem in many cases is that the data needed to make these decisions is not always readily available. And even if it is, it may be quite difficult to aggregate that data remotely in a way where it can be used to make intelligent decisions. The Remote IoT Weather Sensor Station aims to solve these issues by allowing data to be remotely aggregated from devices anywhere in the world, and then presenting a platform where that data can be collected and analyzed, from which intelligent decisions can be made.

The importance of this problem is easily seen in the statistics. There is an incredible amount of energy being wasted every year - totalling $130 billion in homes and business. Heating and cooling are the driving forces of this, which account for 48% of energy consumption in households. There is also a significant amount of water wasted every year for irrigation purposes, as it is estimated that 60% of water consumed for irrigation is wasted[2]. Hopefully, this project can serve to contribute in some way towards reducing these startling statistics.

## II. OVERVIEW

### *A. Hardware:*

The Cypress WICED Wi-Fi CYW43907 Evaluation Kit (CYW943907AEVAL1F) is the main piece of hardware used in this system. The main component on this board is a Murata certified module hosting Cypress' CYW43907 Wi-Fi chipset. These certified modules are incredibly popular amongst IoT developers, as the use of these devices avoids all of the complex and costly RF work and certifications generally required for such products, drastically reducing the time to market. We also connected a PSoC AFE Shield to the kit, which has all of the pertinent sensors for collecting the data we need.

[1] "Welcome to My Smart Home: The 12 Best Devices to Make Your House Smart." Fortune, Fortune, www.fortune.com/2017/02/17/smart-home-tech-internet-of-things-connected-home/.

[2] "Spooky Statistics About Energy And Water Waste." Www.erc-Co.org/, 13 July 2016, www.erc-co.org/spooky-statistics-about-energy-and-water-waste/

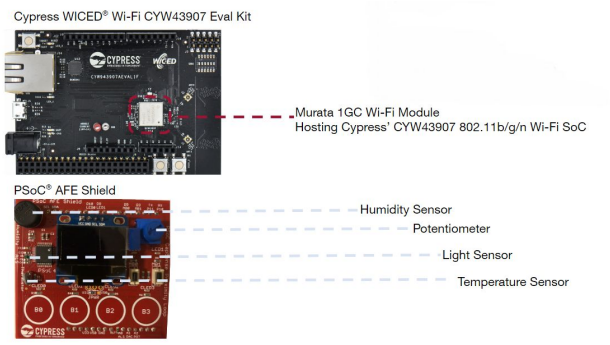


Fig. 1: Hardware Overview

The PSoC AFE shield features a variety of different weather-related sensors measuring humidity, light, and temperature, as well as a potentiometer which could be used for remote manual control and administration of a system[3].

For example, humidity data collected from such a device would be useful in smart garden applications, and could be used to automate timing and quantity when watering crops and other plants. On the other hand, temperature data could be provided to a smart thermostat like Nest, or independently leveraged to create a customized indoor climate system that suits the users wants and needs. Meanwhile, luminosity data would be critical for smart home applications in determining current brightness and then acting on that information, perhaps compelling a set of smart blinds to change their angle relative to the sun's natural light in order to save energy. Finally, potentiometer readings would be ideal in developing a system that could be remotely controlled from anywhere, offering a method of manual control whenever the automated controls don't work out in a user's favor. With regards to the applications we're focusing on for this project, the potentiometer could easily be used to control an actuator that releases additional or less water depending on the variable resistance.

*B. Software:*

A variety of different tools and platforms were used during the creation of this system, with perhaps the most important one being WICED Studio, which is Cypress' Software Development Kit (SDK) for IoT applications. This tool provides a comfortable workspace and a collection of IoT-relevant developer tools. More importantly however, it contains a suite of compilation and flashing capabilities customized for the Cypress evaluation board.

The program used to control the board was written in C. It includes calls to numerous Cypress IoT libraries, which provided functionalities that were essential to all of the board's functionalities. For example, the libraries made it easy for us to MQTT as the main communication protocol in this system, and to also secure those communications using AES encryption.

In order to actually display the data, we used a variety of web technologies such as HTML, Javascript, and a number of Javascript libraries such as Chart.js, Aes-js, and Paho-MQTT to create a web application that could automatically retrieve weather information from IoT devices, decrypt it on the spot, and then display the information over time in the form of vibrant and informative line graphs.

Finally, we created an instance of an Amazon AWS EC2 Linux server to ensure that all of these things could be come together and be used from anywhere in the world[4]. To do this, we used httpd to host our web application and make it accessible via any web browser, and ran mosquitto as an MQTT broker to serve as the center of all of the Remote IoT Weather Sensor Station's communications.

[3] Cypress Semiconductor. CYW943907AEVAL1F Evaluation Kit User Guide. CYW943907AEVAL1F Evaluation Kit User Guide, Cypress, 2017.

[4] AWS. "Amazon Elastic Compute Cloud Documentation." Amazon Elastic Compute Cloud Documentation, User Guide for Linux Instances, www.aws.amazon.com/documentation/ec2/.

## III. SYSTEM DESIGN

### *A. Web-Application (MQTT subscriber)*

*Communications:*

As mentioned before, users can easily access the web application from anywhere by simply requesting a web page from the HTTP server running on the AWS EC2 instance using their web browser.

The webpage received is initially blank, and as soon as it is loaded the page uses the Paho-MQTT Javascript library to *become an MQTT client (a subscriber)*. It then connects to the mosquitto instance (the MQTT broker) on the AWS EC2 server via WebSockets and subscribes to the topic "Stats".

The code for this process can be found in `display_weather_data/index.html`. It's also worth mentioning that, in order to connect to another server hosting an MQTT broker like ours, this file must be modified to connect to the IP address of said server. Furthermore, if mosquitto is used as the MQTT broker on that server, then it must be configured to use WebSockets. This can be done by adding

```
listener <some_port_#>
protocol websockets
```

to the mosquitto configuration file. The port number here must also match the port number used in `display_weather_data/index.html` to connect to the MQTT broker[5].

Assuming that everything is properly configured, the webpage is then effectively ready to function as an MQTT subscriber, and is able to receive and display device data.

[5] Eclipse, Mosquitto. "Eclipse Mosquitto™ An Open Source MQTT Broker." Eclipse Mosquitto, Documentation, www.mosquitto.org/

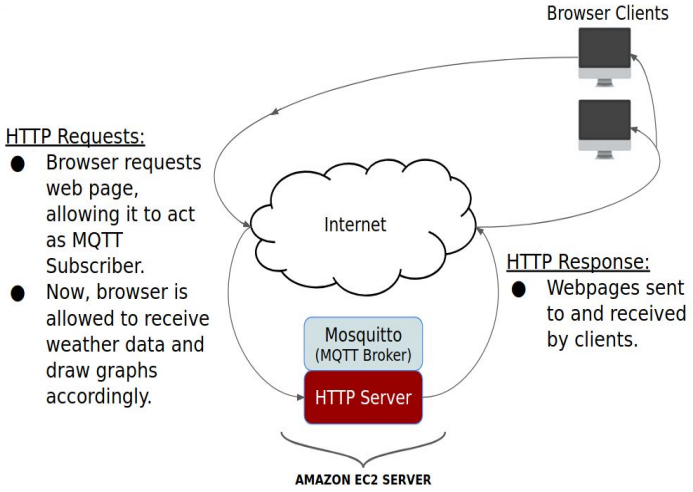


Fig. 2 : Overview of HTTP architecture and generation of MQTT web application subscribers.

### *B. Cypress CYW943907AEVAL1F (MQTT publisher)*

*Communications:*

Meanwhile, the Cypress evaluation kit and the PSoC AFE Shield perform all of the heavy lifting in actually gathering the data and then streaming it to the AWS EC2 cloud server. In terms of communications, the system simply connects to a Wi-Fi network, and then repeatedly extracts the sensor data from the PSoC AFE Shield and publishes that data to mosquitto, our MQTT broker in the EC2 server. The broker then sends these messages to our web client subscribers where it is then graphed, so that remote users (and potentially intelligent systems) can see this information and make informed decisions using this data.

And in order to get the board connected to a local Wi-Fi network so that it can communicate with the remote MQTT broker, a few setup steps must be taken. In order for the board to connect to an access point, it's SSID (and passphrase, if applicable) must be specified by the board for a successful connection. This can be done with our project by navigating to the file `gather_weather_data/wifi_config_dct.h` and changing the `CLIENT_AP_SSID` field to the SSID of your local access point, in addition to changing the

`CLIENT_AP_PASSPHRASE` field to the passphrase of said access point (if applicable).

Furthermore, the application running on the board, whose source code can be found in the file `gather_weather_data/gather_weather_data.c`, must be aware of the MQTT broker address and must also have a custom MQTT ClientId, which can both be specified by setting `MQTT_BROKER_ADDRESS` and `CLIENT_ID` respectively in that file. Other similar options for the board can also be set or modified in that file.

Once all of these options are set, the program is ready to run and can be flashed on to the Cypress CYW943907AEVAL1F board using WICED Studio. Configure the make target for the application and flash it onto the board.

The program will first connect to the Wi-Fi router and the MQTT broker (using the options we just set above) and enters a loop that repeats every 1 second. First, the board will retrieve temperature, humidity, light, and potentiometer data as a single message from the PSoC AFE Shield using I2C. Then the data is extracted and placed in a string, formatted in a way that the web application can understand. Each line in the message is formatted like so:

```
<Field>:<Value>
```

And an example message is shown below:

```
ID:IoT_Device_1
Temperature:50
Humidity:20
Light:1000
POT:1.0
```

Each message must always start with an ID, which is used to create a group of graphs with the web app. The ordering of the lines after this does not matter.

We decided to include the ID in the message (as opposed to just extracting the MQTT ClientId field from it), as this method could be used to allow one device to create multiple groups of graphs on the web application (by having one board send out messages under multiple ID's) or to have multiple devices all work together to create a single group of graphs (by having multiple boards send messages under a single ID). This isn't the way our project operates, but we felt that sending messages using this format would allow for interesting future applications and uses.

Finally, the message is published and sent to the MQTT broker under the topic "Stats", and then the loop repeats. The MQTT broker simply forwards the data to all web application instances (MQTT subscribers to the topic "Stats") where it is then be graphed and displayed to users. An overview of this communications process is shown below:

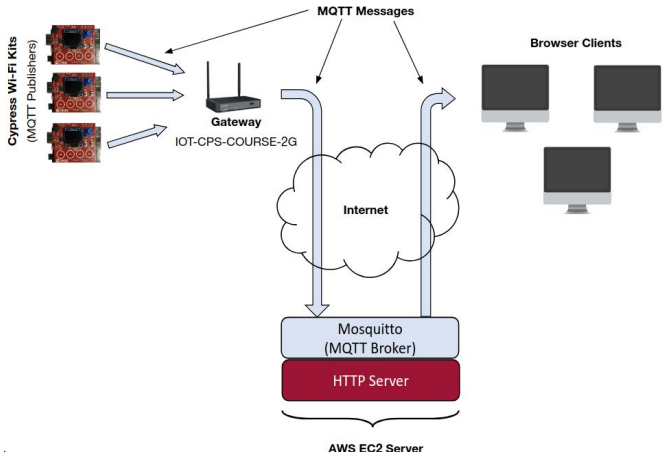


Fig. 3: Overview of MQTT architecture and interactions between IoT weather station devices, remote MQTT brokers, and web application clients.

*C. Encryption:*

However, when weather information is transmitted from the IoT boards to web clients, it is sent as ciphertext using AES encryption. And in order to encrypt messages as efficiently as possible (in terms of power consumption and performance), we utilized the elliptic curve cryptography hardware engine found on the Cypress CYW943907AEVAL1F board. All messages are encrypted using a 128-bit AES key on the board, and are then decrypted using

the Aes-js Javascript library once they're received by instances of our web application.

This method ensures that attackers are unable to look into any messages sent, and are also unable to attack the system by sending false or fake messages. If they attempt to send plaintext or false ciphertext messages to the MQTT broker, the messages will simply "decrypted" into a junk string by recipient web clients and be discarded. And so long as the AES key stays secret, there will be no way for the attacker to send a malicious encrypted message in an attempt to fool or trick the system, nor will they be able to look into any of the data sent.

An overview of our encryption scheme and how it fits into the system is shown below:

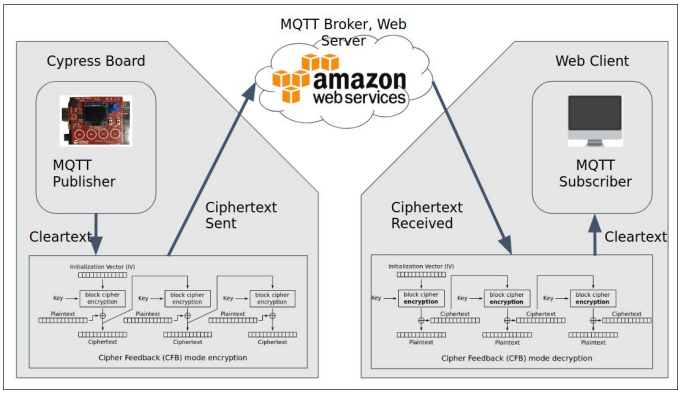


Fig. 4: Overview of encrypted message sending

*D. Displaying the Data:*

Once the data reaches the instances of our web application and is decrypted, it is parsed, fragmented and stored in one of multiple data structures depending on the type of the data and the device the data is from. Immediately afterwards, the web application uses these data structures to construct (or reconstruct, if they are already on the screen) graphs with the Javascript library Chart.js, effectively showcasing the data in a simple, clean manner which can then be used for one of the many prominent applications outlined in the introduction section of this paper.

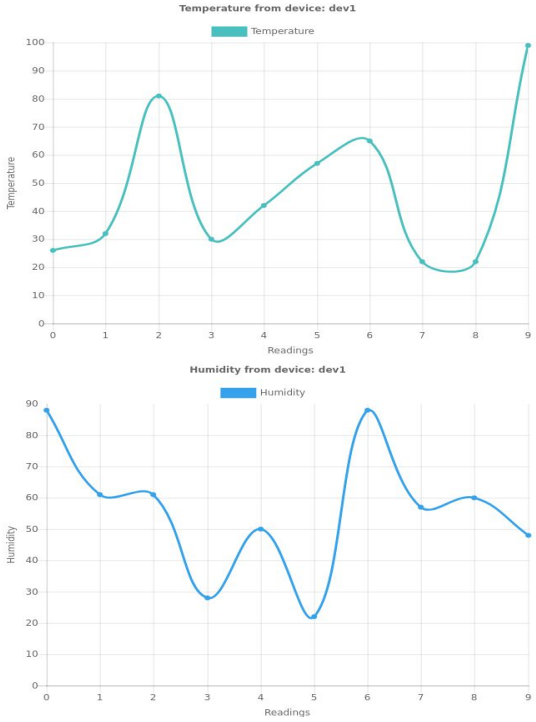


Fig. 5: Example charts displaying data in the web application

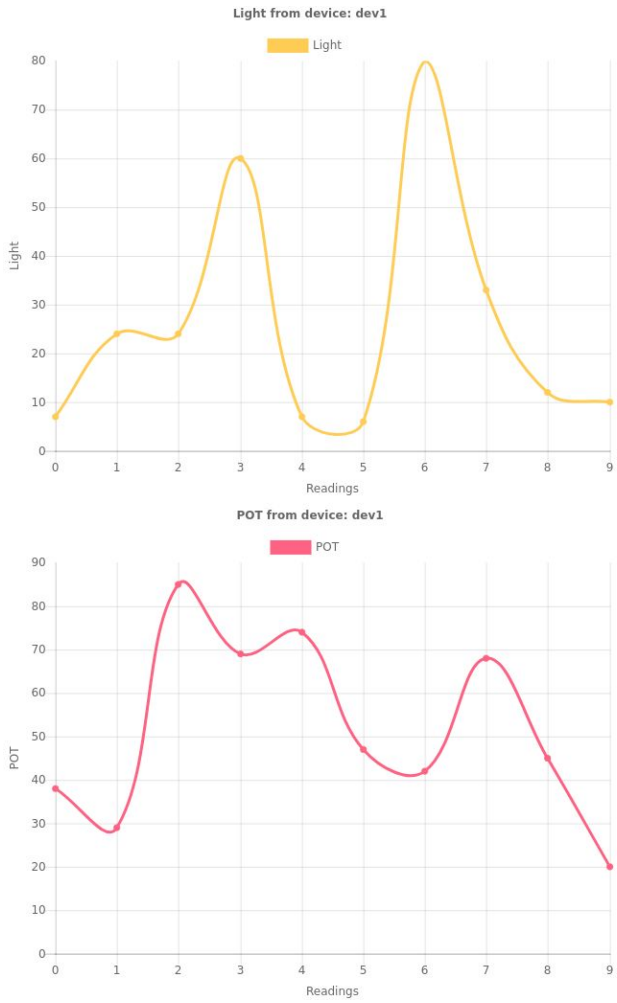


Fig. 6: Example charts displaying data in the web application

## IV. EVALUATION AND CONCLUSIONS

To validate our system design and provide proof of concept, we put it through a series of tests as follows:

A. Connection Test
B. Temperature Test
C. Humidity Test
D. Potentiometer Test
E. Luminosity Test

*Connection Test*:

In order to perform the connection tests, we performed the following procedure:

1) Run the publisher code on the boards to specify MQTT broker address and local WiFi.
2) View webpage to ensure data is being sent and displayed from both devices.
3) Disconnect one device to show how it affects the user interface of the webpage.
4) Disconnect both devices.
5) Reconnect both devices, and observe the displayed data.

*Connection Test Results*:

In performing the procedure above, we noticed that the boards have a variation in the time they take to connect to the server. For example, at step 5 we connected board A and board B sequentially with minimal time between, yet board B connected before board A. However, this had absolutely no negative effect on the results of the experiment and there were generally no problems connecting as long as the Wi-Fi signal was strong enough.

*Temperature & Humidity Tests:*

In order to perform **both** the temperature and humidity tests we performed the following procedure:

1) Connect three of the devices to simulate three separate systems.
2) Once devices have established connection to the EC2 server, breathe on the thermistors and humidity sensors located on the boards.
3) Ensure data transmitted to the EC2 server is being displayed properly through the user interface.

*Temperature & Humidity Test Results*:

In performing the procedure above, we were able to sense and display a change in temperature from an ambient environment to a warm environment as a result of human breath. We were also able to sense and display the change in humidity levels between an ambient environment and that of a human's breath. Between the temperature and

humidity tests, we were able to demonstrate proof of concept for transmitting and displaying real time temperature and humidity data from a remote location with changing environments.

*Potentiometer Test:*

In order to perform the potentiometer test we came up with the following procedure:

1) Connect two of the devices to simulate two separate systems.
2) Turn the knob right and left on the potentiometer to change the voltage potential.
3) Ensure data transmitted to the EC2 server is being displayed properly on the user interface.

*Potentiometer Test Results:*

In performing the procedure above, we were able to demonstrate proof of concept for transmitting and displaying real time potentiometer data from a remote location.

*Luminosity Test:*

In order to perform the luminosity test we drafted the following procedure:

1) Connect three of the devices to simulate three separate systems.
2) Use two flashlights and pan the lights over two devices (one light per device).
3) Simultaneously, cover the third device to minimize the exposure to light.
4) Ensure data transmitted to the EC2 server is being displayed properly on the user interface.

*Luminosity Test Results*:

In performing the procedure above, we were able to sense and display a change in luminosity between an environment with a typical luminosity level and an environment with significantly more intense light levels. We we were able to demonstrate proof of concept for transmitting and displaying real time luminosity data from a remote location with changing environments.

## V. FUTURE POTENTIAL

Development on additional hardware platforms is inherently supported. MQTT is data-centric so technically any type of IoT board with Wi-Fi capabilities will work; as long as the data that is sent over MQTT messages is encrypted and formatted correctly.

Further potential applications that the Remote IoT Weather Sensor Station can service are home climate control and automated smart gardens. These could easily be added to our modular framework as MQTT subscribers, and use the same data that our web application receives in order to make intelligent decisions based on climate data. In addition, support for smart farming is another potential application area as it incorporates many of the same types of weather sensors that our web application currently receives.

The applications discussed above as well as others could be further serviced by adding support for different sensor types. Barometric pressure sensors for predicting potential rainfall is one example of a useful sensor type that could add value to certain applications. Adding new types of fields for new types of sensor data would also be easy, as the web-application is quite modular, and doing so would only amount to adding an extra few lines of code across the following 2 files:

- `display_weather_data/js/config.js`
- `display_weather_data/js/update.js`

There is potential value added in enabling GPS location services to the Remote IoT Weather Sensor Station as well. This could be used periodically by the web application and referenced versus locational data online to check if sensor operation is still accurate. Intelligent insights can only be determined by reliable data and so device

health is very important. Thus locational services would be a further beneficial development and remove the cumbersome need for users to manually check individual devices.

Development of a mobile application would be favorable for user experience as well. Browser clients offer robust services to make large-scale, impactful intelligent decisions off of aggregated weather sensor data, however a mobile application would provide further benefits to users in that they would be able to quickly and effortlessly check the status of their IoT system, from any network in the world.

The Remote IoT Weather Sensor Station's current implementation enables a variety of IoT applications: the capabilities needed to increase quality of life, provide monetary savings to corporations, provide environmental benefits, and in addition, has the potential to be expanded and improved on to service the needs of future IoT applications.

## VII. REFERENCES


1. "Welcome to My Smart Home: The 12 Best Devices to Make Your House Smart." Fortune, Fortune, www.fortune.com/2017/02/17/smart-home-tech-internet-of-things-connected-home/.
2. "Spooky Statistics About Energy And Water Waste." Www.erc-Co.org/, 13 July 2016, www.erc-co.org/spooky-statistics-about-energy-and-water-waste/.
3. Cypress Semiconductor. CYW943907AEVAL1F Evaluation Kit User Guide. CYW943907AEVAL1F Evaluation Kit User Guide, Cypress, 2017.
4. AWS. "Amazon Elastic Compute Cloud Documentation." Amazon Elastic Compute Cloud Documentation, User Guide for Linux Instances, www.aws.amazon.com/documentation/ec2/.
5. Eclipse, Mosquitto. "Eclipse Mosquitto™ An Open Source MQTT Broker." Eclipse Mosquitto, Documentation, www.mosquitto.org/.
6. AWS. "AWS IoT Core Documentation." Developer Guide, AWS IoT, www.docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html.