Cloud Temperature & Humidity Notification System

Lue Xiong

April 26, 2020

Contents

1 Context

The Cloud Temperature & Humidity Notification System is about an IoT system that gives the ability to notify user(s) of temperature and humidity fluctuations within their living environment through the usage of a Simple Message Service, which is also known as SMS. The system will also notify a user if the Particle Argon is offline by checking that data is being sent and stored in the database. Though this is the main concern of the system, it also allows users to visualize their daily climate averages in an interactive graph. The graph is automatically updated for the users to view whenever they want to in the web. Behind the scenes, most computation are abstracted away from the user by using Google Cloud Platform. The initiation of these functionalities start from the temperature and humidity readings being published by the Particle Argon.

Working within the limitations of a small apartment and time alloted for the project, I am unable to realize the full potential of the system. This project represents a single IoT device that enacts the above mentioned functionalities. One can imagine however, being able to send in-home area location data and climate readings with a multiple of these IoT devices scattered across a home with multiple rooms and stories. A user would be notified where in the house and when the climate has reached configured threshold levels for each device. They would then be able to see data points for each specified area of the home. Ideally aspects of the system such as climate data publishing intervals, setting climate threshold and maximum notification intervals, and timezones should be configurable. With that said, the project seeks an minimum viable product. That is, the system will notify users of climate thresholds being met and device status through SMS notification as well as graphing average climate per day for viewing averages over days. User configuration will be beyond the minimum viable product as it would require additional development of an mobile and/or web application.

Though the system is targeted for the home living environment, it can be used for environments that require careful monitoring. Take a fermented product like kombucha for instance; it needs to be fermented in an environment where temperature hovers in the range of 65 to 85 degrees fahrenheit over the course of a week to multiple weeks. Low temperatures will either stop or dramatically slow down the fermentation process. High temperatures will quicken the fermentation process but also increases the risk of unwanted bacteria or mold growth that would ruin the product. Striking a balance with temperature for optimal conditions is difficult without information. The usage is only limited to the imagination.

2 Problem

The problem trying to be solved are giving the user climate data about their living environment. It is clear that certain ranges of temperature and humidity affect humans in ways that are detrimental to their health. For example, low humidity environments –characterized as 30% relative humidity and below – is a condition for being prone to respiratory infections, dry eyes, and itchy irritated skin. High humidity environments – characterized as 60% and above – is a condition for bacterial and mold growth, catalyst for decomposition of organic materials, and attracts bugs and insects to the home. A way to solve that problem is to give a renter or home owner actionable data to understand that their living environment is in need of change. That is where the Cloud Temperature & Humidity Notification System comes in.

3 Market Research

From market researching, there are a couple of products that have similar functionalities:

- AcuRite 01166M 3-Sensor Indoor Monitoring
- Govee Temperature Humidity Monitor
- Proteus AMBIO

All three of these products measure temperature and humidity and have their own value propositions. We'll take a look at what they do and how they differ from this system.

3.1 AcuRite 01166M 3-Sensor Indoor Monitoring

AcuRite 1166M is packaged with 3 sensor units along with what they call a *smartHUB*. The name is self-explanatory, it is the central communication piece for the 3 sensors. The max connection capacity for the smartHUB are 10 sensors. These sensors will read in the climate information of the environment and send it over to the smartHUB, which will then send information over the network to be processed and viewed online. There are some inherent flaws with this system as I will explain.

With the max capacity being 10 sensors connected to the smartHUB, it is only usable in a home environment and even then, a user may want more sensors if they have a larger home. The reason for the limitation is the design of having sensors centered around the smartHUB, and not standalone pieces that can interact with a network. The sensors are nothing without the smartHUB, therefore renders the system useless if it breaks. It also adds one more layer of complexity that is unnecessary for the user. Unnecessary because it is not doing anything complex enough to justify a central hub.

Another issue is reliability. The range at which these sensors can communicate with the smartHUB are limited and depending on the structure, materials, and size of the home, can make it more difficult to reliably transfer information from sensor to hub. This issue of reliability is further enforced by users of the system.

3.2 Govee Temperature Humidity Monitor

3.3 Proteus AMBIO

4 System Overview

This section will describe the minimum viable product requirements and the architectural decision made for achieving the desired results of the system.

4.1 System Requirements

In order to satisfy the need of giving a user useful information about temperature and humidity in their targeted environment, the following are the minimum viable product requirements:

- All notifications sent to the user shall be an SMS message
- Notifications sent to the user shall be based on the following criteria
  - Temperature upper threshold shall be set to lesser than or equal to 75 degrees fahrenheit
  - Temperature lower threshold shall be set to greater than or equal to 65 degrees fahrenheit
  - Humidity upper threshold shall be set to lesser than or equal to 60 percent relative humidity
  - Humidity lower threshold shall be set to greater than or equal to 30 percent relative humidity
- Minimum 1 hour gap in between each SMS message per climate type (AKA temperature and humidity) reaching above or below threshold value
- Check device status every hour
-

4.2 Event-Driven Architecture

The approach taken for the system as mentioned before is heavy on the software side which manifests into an event-driven serverless architecture. An event-driven serverless architecture in the system context allows the climate data readings to be sent as events, processed based on its event type, and have actions performed without the need for standing up servers. The following are benefits of the said architecture:

- Loose coupling between software components
- Automatically scale based on user demand
- All processing driven by event
- No need for management of servers
- Inherent stateless nature
- Overall cost reduction versus managed servers

While these benefits are good to have, every architectural decision has it's drawbacks. The following are drawbacks of the said architecture:

- Each serverless function must provide its own packaged dependencies
- Having large amount of serverless functions can become unwieldy
- Long running processes are not fit for serverless functions
- Handling system state amongst stateless functions can be tricky

Using an event-driven serverless architecture made sense for this particular system, as the needs of the system are fulfilled automatically on publishing data without any manual intervention.

5 Process

I will preface that the Cloud Temperature & Humidity Notification System is heavily software-based. The hardware components are nothing special and ultimately serve as a vessel for sending climate information to be processed in the cloud to create the closed-feedback loop. The Particle Argon referenced in Section 5.1 will contain code that is fundamental to the system requirements and architectural decision in Section 4. This particular section will describe the thought process behind the implementation, considerations taken during development, and how the system performs user notification and automatic graph updates.

5.1 Hardware Components

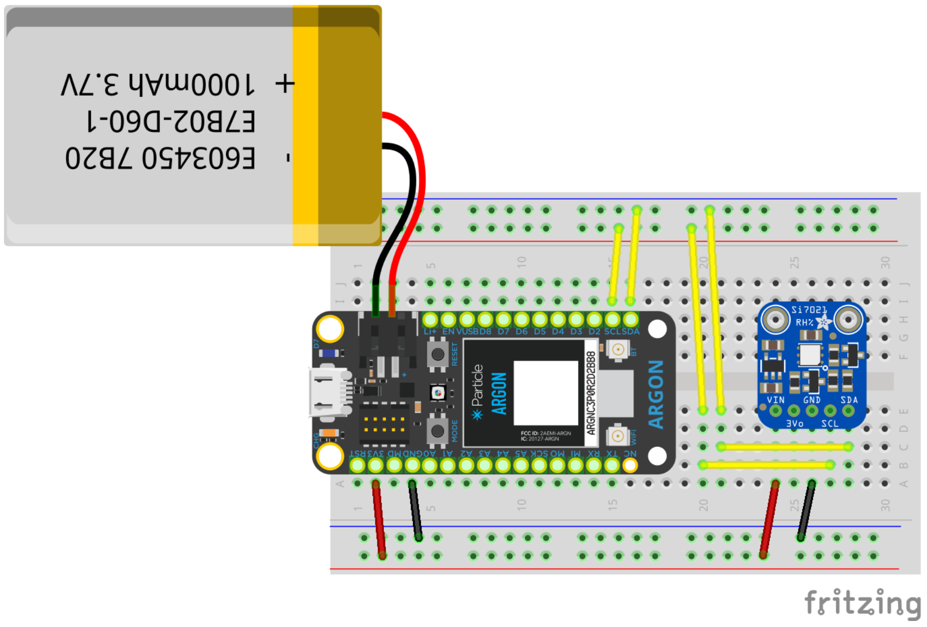

Figure 1: Breadboard Schematic

Figure 1 consists of the following hardware components:

- Particle Argon – IoT development kit for connecting and sending information over the network
- Adafruit Si7021 – Temperature and humidity sensor breakout board
- Breadboard – Base to create circuits and prototype electronics
- Jumper Wires – Cables that are able to connect components
- Lithium Ion Polymer Battery 1000 mAh – A power source for the conntected components

5.2 Implementation with Particle Argon & Particle Cloud

```
void setup() {
  Time.zone(-5);
  sensor.begin();
}
```

Figure 2: setup() function

The **setup()** as shown in Figure 2 is important here for two reasons. The first part is that the Particle Argon is in default timezone of Universal Time Coordinated (UTC) which differs from the timezone I live in is Central Daylight Time (CDT). To be able to represent the timezone difference, setting the default timezone with an offset of minus 5 hours is required. The second part is making sure that the Adafruit Si7021 is enabled after being initialized because it is the central piece for reading temperature and humidity from the targeted environment.

```
void loop(void) {
  int degreesCelsius = sensor.readTemperature();
  int degreesFahrenheit = (degreesCelsius * 9 / 5) + 32;
  int relativeHumidity = sensor.readHumidity();

  Particle.publish("temperature", String(degreesFahrenheit),
      PRIVATE);
  Particle.publish("humidity", String(relativeHumidity), PRIVATE);

  if (isEndOfDay() == true) {
    Particle.publish("climateAverageUpdate", "Updating␣climate␣
        data␣file", PRIVATE);
    publishIntervalMilliseconds = 1800000;
  } else {
    publishIntervalMilliseconds = 900000;
  }

  delay(5000);

  System.sleep(systemSleepConfiguration
    .gpio(WKP, RISING)
    .network(NETWORK_INTERFACE_CELLULAR)
    .flag(SystemSleepFlag::WAIT_CLOUD)
    .mode(SystemSleepMode::STOP)
    .duration(publishIntervalMilliseconds));
}
```

Figure 3: loop() function

In Figure 3, the **loop()** function shows that temperature and humidity are being read with the sensor object. They are then used as values to publish to the Particle Cloud. From that point on they are passed to Google Cloud Platform. However, there will be further explanation on how this integration works and what processes happen on the Google Cloud Platform in Section 5.3.

```
bool isEndOfDay() {
  int now = Time.now();
  int currentHour = Time.hour(now);
  int isMorning = Time.isAM(now);

  if (updated == false && currentHour >= 21) {
    updated = true;
    return true;
  }

  if (updated == true && currentHour >= 6 && isMorning) {
    updated = false;
    return false;
  }

  return false;
}
```

Figure 4: isEndOfDay() function

Beyond publishing the climate values, the **loop()** function checks whether the day has ended from the perspective of the system. The end of the day is defined as code in Figure 4. Before it was mentioned that a UTC offset of minus 5 hours was applied in the **setup()** function, and this is where it comes into context. The end of day is 9 PM CDT, which is equivalent to 21 in a 24 hour format. Once the system understands that it is the end of the day, an event is published to update a file containing daily climate average aggregates for the day. Same as before this will be further explained in Section 5.3.

In relation to the end of the day, a variable called **publishIntervalMilliseconds** will be set to 1800000, in other words, 30 minutes. This publishing interval will last for 9 hours starting from 9 PM CDT to 6 AM CDT. During the working day, the **publishIntervalMilliseconds** is set to 900000, or 15 minutes. This plays a role in conserving battery because during the night, it is only publishing climate data twice per hour as opposed to four times per hour. However, the bigger battery saving impact comes from placing the Particle Argon in a sleeping state.

A sleep state simply means that the Particle Argon turns off all unnecessary internal processes that consume energy and maintains a low-power state for a duration of a configured time. There are two main modes of sleep state that Particle Argon can go into: **STOP** and **HIBERNATE**. The decision was to use the **STOP** sleeping mode because it provides the opportunity to keep the network connection alive while it is in a low-power state. **HIBERNATE** does not allow this, and if it was used, it would require the Particle Argon to completely establish a new connection every time it wakes up from the specified duration of **publishIntervalMilliseconds**; defeating the point of using it.

5.3 Implementation with Google Cloud Platform

5.3.1 Purpose of Tools Within Platform

5.3.2 SMS Notification

5.3.3 Climate Data Graph

6 Discussion of Results

7 Conclusion