Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE

A Distributed System for Supporting Smart Irrigation using IoT Technology

Ahmed Abdelmoamen Ahmed*, Suhib Al Omari[1] | Ripendra Awal, Ali Fares[2] | Mohamed Chouikha[3]

[1]Department of Computer Science Prairie, Prairie View A&M University, TX, USA
[2]College of Agriculture and Human Sciences (CAHS), Prairie View A&M University, TX, USA
[3]SECURE Center of Cybersecurity, Prairie View A&M University, TX, USA

Correspondence
*Corresponding Ahmed Abdelmoamen Ahmed, This is sample corresponding address. Email: amahmed@pvamu.edu

Summary

In this paper, we present the design and implementation of a smart irrigation system using Internet of Things (IoT) technology, which can be used for automating the irrigation process in agricultural fields. It is expected that this system would create a better opportunity for farmers to irrigate their fields efficiently, as well as eliminating the field's under-watering, which could stress the plants. The developed system is organized into three parts: sensing side, cloud side, and user side. We used Microsoft Azure IoT Hub as an underlying infrastructure to coordinate the interaction between the three sides. The sensing side uses a Raspberry Pi 3 device, which is a low cost, credit-card sized computer device that is used to monitor in near real-time soil moisture, air temperature and relative humidity, and other weather parameters of the field of interest. Sensors readings are logged and transmitted to the cloud side. At the cloud side, the received sensing data is used by the irrigation scheduling model to determine when and for how long the water pump should be turned on based on a user-predefined threshold. The user side is developed as an Android mobile app, which is used to control the operations of the water pump with voice recognition capabilities. Finally, this system was evaluated using various performance metrics, such as latency and scalability.

KEYWORDS:
Irrigation; IoT; Soil Moisture; Sensors; Azure; Android

1 | INTRODUCTION

In the United States, landscape irrigation consumes around 34 million m^3 each day [1]. A significant percentage of that water use is wasted due to overwatering caused by inefficiencies in traditional irrigation methods and systems. To reduce the wasted water, it is becoming increasingly important to optimize the agricultural irrigation methods using advanced technologies such as Cloud Computing [2], Remote Sensing [3] and Internet of Things (IoT) [4], which are used to gather data from various sources in the field for enhancing predictive decisions. In the agriculture field, sensing capabilities for quickly and inexpensively generating agriculture and food cyberinformatics have improved immensely in the past few years [5].

Imagine a smart IoT system which measures the spatial variability of soil properties in agricultural fields, monitors farm conditions, and plans irrigation. Such applications would tackle production costs and operational challenges for both small- and large-scale farmers. These applications rely on the state of the context in which sensing devices are located, such as geographical location, proximity, temperature, wind speed and direction, solar radiation and humidity. [6]. Increasingly, sensed data could also inform decisions to activate actuators to carry out tasks automatically[7]. A growing number of smart farming technologies offer good examples of such capability.

The objective of this work is to develop an IoT smart irrigation system based on a near real-time monitor of soil moisture in the plant root zone[1]. We developed a distributed system which is organized with parts executing on IoT sensing devices, on the cloud, as well as on the user devices. The communication between the three sides is coordinated through Azure IoT Hub[8], which is a cloud-based platform that enables a tremendous number of IoT devices to communicate between themselves in a scalable and reliable manner.

At the sensing side, we used soil moisture, air temperature, and relative humidity sensors to monitor the current moisture in the soil, air temperature, and relative humidity in the field, respectively. We developed a sensing-side application, hosted on a Raspberry Pi 3 device, which receives the sensed data from these sensors, preprocess it, and send the processed feeds to the cloud side. Also, the Raspberry application enables farmers to manually control the irrigation process by turning on/off a water pump based on the current moisture level of the soil, which is also displayed on the Raspberry application.

At the cloud side, the Microsoft Azure platform has been used as a central bidirectional communication IoT hub among the system components. The hub supports multiple forms of messages to keep track of the current state of the IoT end-devices such as cloud-to-device telemetry messages for controlling the devices remotely. These telemetry messages are sent durably to accommodate intermittently connected devices. All IoT devices at the sensing side –including the moisture sensor, temperature sensor, LED, and Raspberry Pi– must be registered as end-devices at IoT Hub Device Provisioning Service in order to send/receive telemetry messages.

At the user side, we developed a desktop application which enables users to remotely control the farm irrigation and lighting devices, customize the automatic irrigation process, and communicate with the Raspberry Pi application. It also gives users the ability to set predefined thresholds for the soil dryness alert property, which warns the user if the moisture level drops to a certain degree. We also developed an Android mobile app for increasing the user experience while using the system. The app has an option to control the irrigation and lighting devices using voice recognized-commands.

The rest of the paper is organized as follows: Section 2 presents related work. Sections 3 and 4 present the design and prototype implementation of the irrigation system, respectively. Section 5 experimentally establishes the performance cost of using the system. Finally, Section 6 summarizes the results of this work.

2 | RELATED WORK

Conventional agriculture is slowly changing towards precision agriculture[9], which is a farming management concept based on observing, measuring, and responding to the spatiotemporal variability in weather, soil, irrigation/water, and agricultural production. The use of intelligent agricultural IoT applications[10,11,12,13,14], through a large number of end-devices in the target areas —such as farmland, greenhouses, forest gardens, pastures-–, which can collect data about agricultural breeding or planting in real-time is becoming increasingly important in modern agriculture. This section focuses on existing work that supports smart farming practices using IoT technology.

Since landscape irrigation consumes billions of gallons of fresh water daily worldwide, tremendous research efforts have been done to make the irrigation process and water usage more efficient. For instance, in[12], the authors used low-cost sensors to implement an automated system for crop field monitoring. Arduino was used to sending the sensed data to a web server, which stores the current values of moisture, humidity, temperature, and light intensity in a database. An Android mobile app was built to enable users to monitor the status of their crop fields.

An IoT-based irrigation monitoring system is proposed in[15], which uses a wireless sensor network and Arduino technology for sensing soil moisture level, temperature and relative humidity values in the agricultural field. The system monitors the water level of the irrigation tank via a water level sensor so that if the water level is below a certain threshold, then irrigation will not start. The sensed data are preprocessed using two Arduino nodes; then the processed data is sent to a cloud server via a ZigBee transceiver and relay switching unit. The cloud server is responsible for making the irrigation decisions based on a set of predefined thresholds. An android application was also developed to notify the user with any change that needs immediate actions such as a sudden rise in temperature, new watering requirements for some species of plants, etc.

[1] Available online: https://github.com/ahmed-pvamu/Smart-IoT-Irrigation-System

Sales et al.[16] proposed a Wireless Sensors Actuators Network (WSAN) system for monitoring and assessing plants water needs. The system consists of three main components: a WSAN, cloud platform, and web application. The WSAN is deployed in the agricultural field to collect soil moisture data. The WSAN has several nodes connected using cluster-tree topology for improving the system scalability. Each node is equipped with a soil moisture sensor. Sensor feeds are collected by a central Access Point (AP) which sends the aggregated data to the cloud server. The Cloud Platform is responsible for validating, processing, and storing the received sensor feeds from the sensing side. Besides, the authors developed an algorithm for controlling the irrigation process based on the collected soil moisture from the WSAN nodes, as well as weather forecasted data from Weather Underground service[2]. This service gets the predicted weather data for the closest available weather station to the location of the field where the sensor nodes are deployed. The algorithm uses the Probability of Precipitation (PoP) value of the target region within 6 hours to decide when the irrigation should start and for how long. When it is the time to spray the field, sensor nodes are instructed by the cloud platform to increase the soil moisture sampling frequency, which allows optimizing the irrigation process because more information is collected. The default sampling rate is restored after watering the field. Finally, a web application is used to show the location of each sensor node, the connected battery status, and data history.

Another platform for managing the components of a precision irrigation system is presented in[9]. The proposed distributed system includes a server node which hosts a decision support system, a mobile application for user interaction, and IoT devices that operate linear irrigation machines. The decision support system creates an irrigation map, which represents the amount of water to be supplied in each cell of the field based on several factors such as the integrating geographic, meteorological, and soil data. An Unmanned Aerial Vehicle (UAV) –equipped with a vision sensor– was deployed to perform an aerial survey over the field to provide a high-resolution measurement of the current state of the field.

An IoT-based smart irrigation system based on machine learning is proposed in[13]. The proposed system aims to achieve optimum water-resource utilization in the field by using a machine learning model, which can predict the irrigation requirements of a field using the sensing of several parameters such as soil moisture and the weather forecast data. The machine learning model uses the sensors' feeds of the recent past few days, and the weather forecasted data for predicting the soil moisture for the upcoming days. However, the proposed system depends entirely on the accuracy of the predicted soil moisture, which is affected by numerous environmental variables such as air temperature, air humidity, soil temperature, etc.

In summary, most of the existing work focuses on narrow application areas or specific concerns, making it difficult to utilize them for a broader class of functionalities. Furthermore, none of these systems implemented a fully-functional ecosystem for agricultural applications starting from collecting data at the sensing side all the way to visualizing processed information at the cloud side. In this paper, we present a complete IoT system which can be used to monitor and control the agricultural field operations.

3 | SYSTEM DESIGN

As illustrated in Figure 1, the distributed run-time system for the irrigation system is organized with parts executing on sensing IoT devices, Azure cloud platform, as well as user devices. In the rest of this section, we discuss these three parts separately.

3.1 | Sensing side

At the sensing side, data can be collected from a variety of IoT end-devices including soil moisture, temperature, rainfall, wind speed and direction, solar radiation, humidity, leaks monitoring, accelerometer, GPS, proximity, motion, and dew point sensors. A Raspberry Pi device – which acts as an IoT gateway– is used to aggregate these data and coordinate the connectivity of the end-devices to each other and to the cloud side. Accurately, the gateway keeps aggregating the received sensor data until a sufficient number of them have been received to detect an interesting event such as a change in the level of soil moisture in an agricultural field. Gateways either send updates periodically or when they observe a new event, to the IoT hub at the cloud side through the device provisioning service. The IoT hub sends a set of parameters to the gateway advising it on how to detect events, construct their messages, and how often to send them (once or periodically, how frequently, etc.).

The IoT Hub Device Provisioning Service is a helper service for the IoT Hub that enables zero-touch real-time provisioning of IoT end-devices. All IoT end-devices must be enrolled with a Device Provisioning Service instance by sending a registration

[2]Weather Underground is a third-party online weather service.

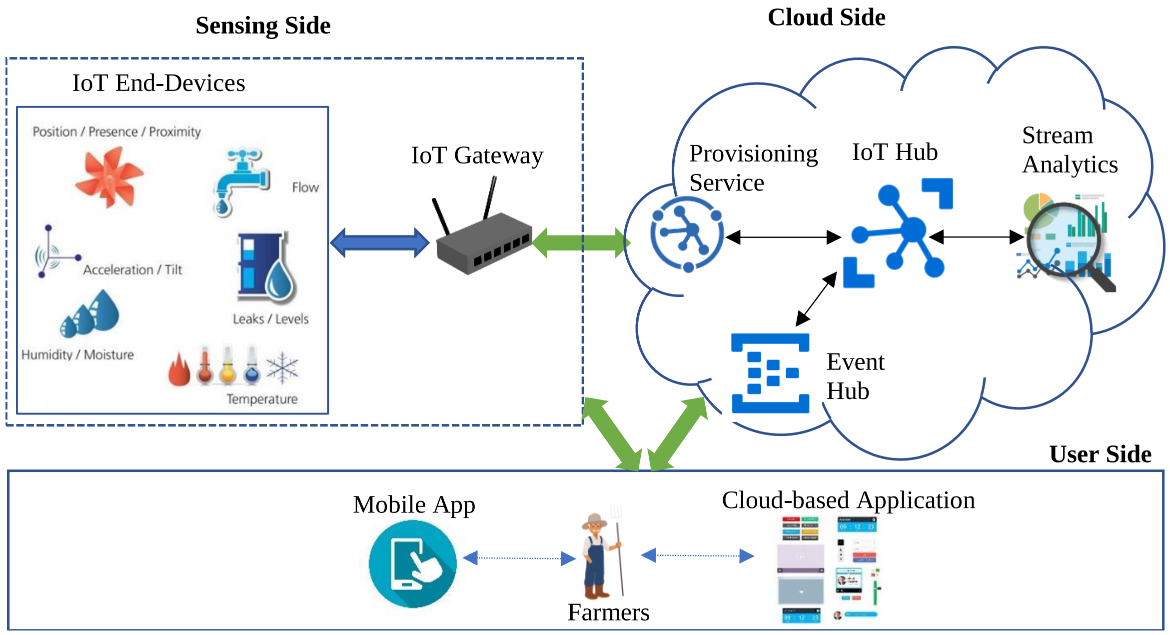

FIGURE 1 System Architecture

request to the service. Once the device has been provisioned, it can boot up, and call the provisioning service to be recognized and assigned to an IoT hub.

3.2 | Cloud side

At the cloud side, the IoT Event Hub [17] is used to receive and aggregate the events sent from the gateway at the sensing side. The Event Hub is a streaming service that is capable of collecting and processing millions of events contained in telemetry messages produced by IoT end-devices. The Event Hub also implements some security mechanisms to ensure that the incoming telemetry messages are legitimate. Event Hubs enqueue the received messages in a partitioned consumer model in which each consumer application only reads a partition of the message stream. This model enables horizontal scale for event processing that can be easily integrated into the big data and analytics services of Azure, including Databricks, Azure Stream Analytics, etc.

Stream Analytics is a serverless event processing engine that can be used to analyze data streams generated from IoT end-devices in real time. The Stream Analytics is employed to implement our automatic irrigation algorithm by detecting the pattern of the soil dryness. Specifically, the analytics service collects aggregated events until a sufficient number of them have been received (as determined by a sufficiency condition) and then triggers actions such as creating alerts, feeding information to a reporting tool, or storing transformed data for later use.

An Azure Stream Analytics job consists of an input, a transformation query, and an output. The events sent from the sensing devices are considered the input source for a job. The transformation query, which is based on SQL query language, is used to aggregate the streaming sensor data to produce the actions which are considered the output of the job.

To control our sensing devices connected to the IoT hub remotely, we used a cloud-to-device interaction model by invoking the direct methods on the IoT end-devices. Direct methods represent a synchronous request-reply interaction with an IoT hub and a sensing device. For instance, direct methods can be used to send an action message to control a water pump in the agriculture field.

3.3 | User side

Non-technical users (e.g., farmers) can easily monitor and control the agricultural field conditions from anywhere with the help of various sensors and actuators (e.g., light, humidity, temperature, soil moisture, etc.). We developed a Graphical User Interface (GUI) which can be accessed from personal computing devices such as PCs and smartphones. The GUI will help users to access

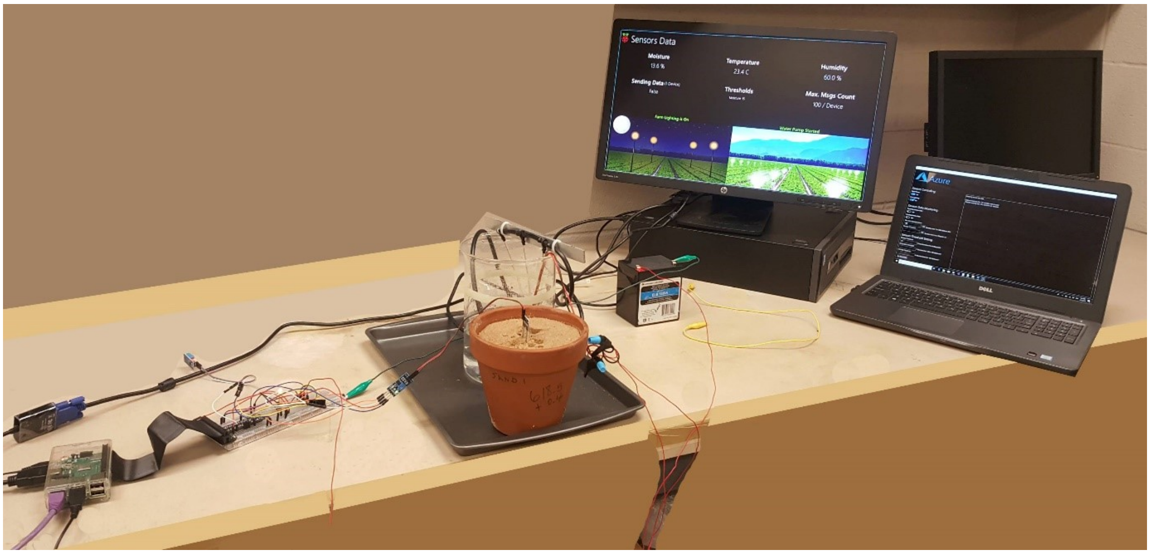

FIGURE 2 The Physical Implementation of the Smart Irrigation System

the deployed IoT system remotely, which will eliminate the need for constant manual monitoring. This design provides cost-effective and optimal solutions for farmers with minimal manual intervention. Furthermore, the GUI can be used to extract real-time insights and actionable information using the Azure Stream Analytics, which would aid the decision-making of both small- and large-scale farmers. This would improve management and crop yields significantly.

4 | SYSTEM IMPLEMENTATION

Figure 2 shows the prototype implementation of the smart irrigation system. Next, we describe the system components at the sensing, cloud, and user side.

4.1 | Sensing side

For controlling the environment in an agricultural field, different sensors that measure the environmental parameters according to the plant requirement have to be deployed in the field. In this project, we tried to remotely control the farm irrigation and lighting devices by using the following hardware components:

- Raspberry Pi 3 Model B+ is used as an IoT gateway at the sensing side for aggregating the sensing data collected from sensors. The Raspberry Pi is equipped with a 64-bit quad core processor running at 1.4GHz, dual-band 2.4GHz, and 5GHz wireless LAN. We installed Windows 10 IoT core on the Raspberry Pi, which can host and run .NET applications. We connected a soil moisture, temperature, and humidity sensor to the Raspberry Pi via the General Purpose I/O (GPIO) pins.

- Water pump equipped with a 12 volt-DC battery is used to circulate the water on an irrigation pot. The pump has two water channels: input and output. The flow of water is absorbed by the input channel and pushed into the bowl through the output channel. We also used a transistor which controls the flow of the electrical current through the circuit. When the transistor receives a signal from the Raspberry Pi to turn the pump on, it allows the electrical current to move through the circuit which turns the pump on, and vice versa.

- Soil moisture sensor is used to measure the current moisture level in the soil. The value measured by the sensor is the electrical resistance of the soil to the flow of electricity between two electrodes. Figure 3 shows an example of the relationship between resistance and water content of the soil moisture sensor.

  The measured value must be calibrated according to the type of soil before converting it to volumetric water content of soil. Precisely, we have calibrated the relationship between resistance and water content of the soil moisture sensor. For our experiments, we assumed the following calibration equation:

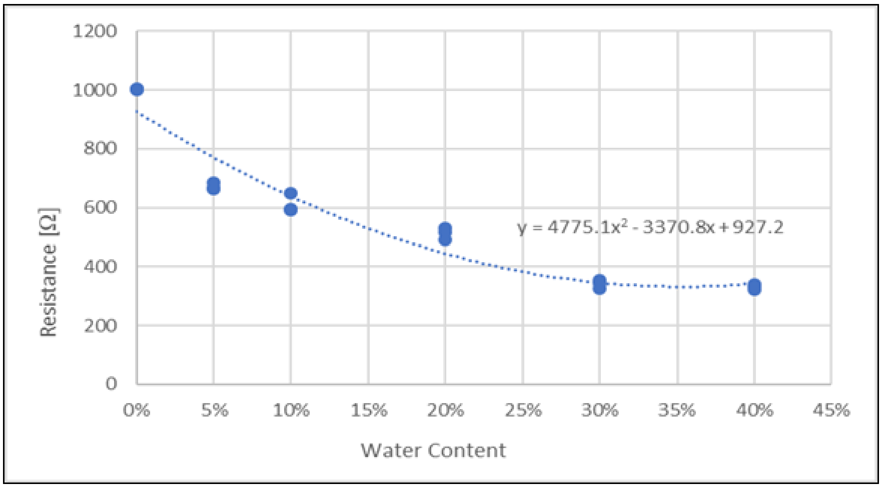

FIGURE 3 The Relationship between Resistance and Water Content of the Soil Moisture Sensor

$$\theta \approx \frac{1{,}023-\delta}{16.575}$$

where θ is the volumetric water content of soil (%) and δ is the soil moisture sensor reading (Ω).

There are three methods of irrigation scheduling: soil-, weather-, and plant-based or combined irrigation scheduling. The latter two methods may need to consider evapotranspiration. In this paper, we measure soil moisture using an in-situ sensor; therefore, there is no need for considering the evapotranspiration in our calculations.

- DH11 Temperature and humidity sensors are used to measure the temperature in Celsius and the humidity in percentage in the field, respectively.
- Light-emitting diode (LED) is used to represent the actual farm lighting, which can be controlled by our system.
- MCP3002 Analog to Digital converter is used to convert the moisture sensor analog reading to a digital value.
- Resistors are used to limit the amount of electrical current moving through the circuit. Particularly, we used one resistor for the LED and another one for the transistor in case one or both of them draw more current than the Raspberry Pi can supply (i.e., around 60mA). In this case, the resistors will ensure that only 60mA will flow through the circuit to protect the connected Raspberry Pi and sensors from damage.
- Jumper wires with two ends connectors are used to connect the Raspberry Pi with all other hardware parts.
- Display screen is connected to the Raspberry Pi via its HDMI port to display the sensing side application.
- Keyboard and mouse are connected to the Raspberry Pi via its USB ports as input devices.
- Breadboard is used to interconnect all hardware components by inserting their terminals or connected jumper wires into the holes of the board.

4.2 | Cloud side

At the cloud side, we used the Azure portal to create an IoT hub instance and two virtual devices connected to the established hub. The first virtual device represents the physical LED, while the second virtual device represents the physical water pump at the sensing side.

As shown in Figure 4, we developed a cloud-based windows application for managing the IoT-end devices at the sensing side. The app uses the IoT hub direct methods to control the devices remotely. On the right-hand side, we display the device-to-cloud telemetry messages sent from the IoT gateway to the Event Hub, and vice versa. On the left-hand side, we build a simple control panel to enable the user to control the end-device at sensing side remotely. The user can perform the following functionalities:

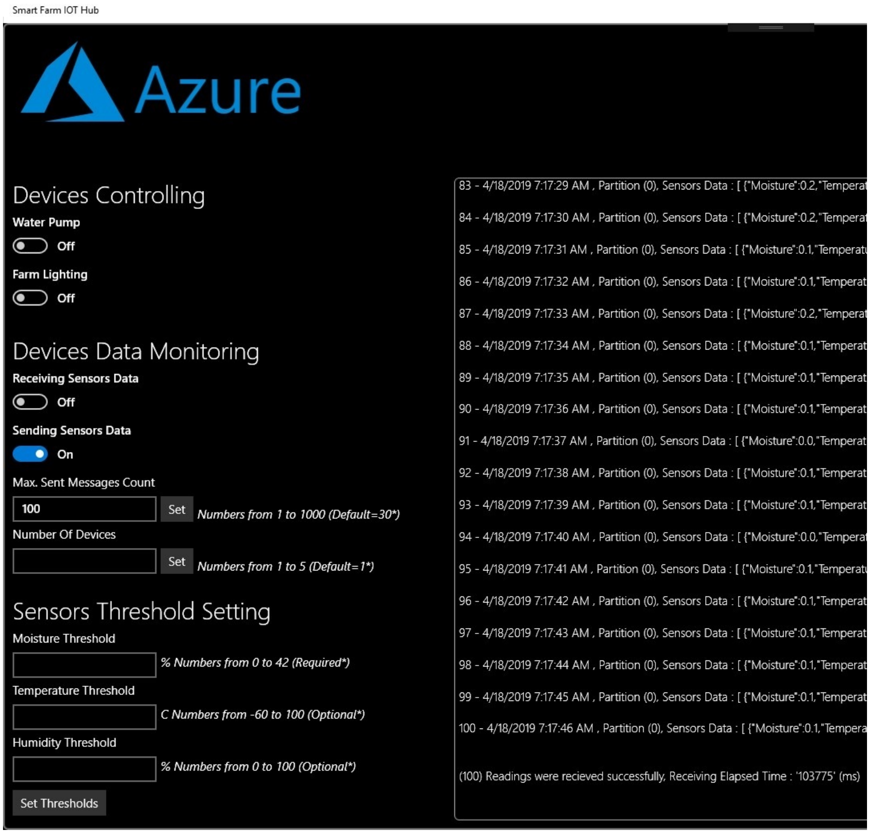

FIGURE 4 The Cloud-based Windows Application for Managing IoT End-devices

(i) turn on/off the LED (e.g., farm lighting); (ii) turn on/off the water pump (e.g., control irrigation); (iii) set the maximum number of telemetry messages which can be sent from any end-device to the Event Hub; and (iv) set the threshold values for the soil moisture, temperature and humidity sensors which are used to fire the dryness alert if immediate attention from the user is needed.

Figure 5 shows a code snippet for receiving and displaying the telemetry messages which are exchanged between the end-devices and the Event Hub.

4.3 | User side

At the user side, we developed a Universal Windows Platform (UWP) application that runs on the Raspberry Pi to display the real-time sensing data from the field (see Figure 6). This application's GUI is designed and developed using XAML[3] and C#, respectively. The application displays the following information:

- The current readings of the soil moisture (in percentage), temperature (in Celsius), and humidity (in percentage) sensors.
- The number of devices which are currently sending sensors readings to the cloud side. It can be remotely adjusted in the cloud-based windows application. We also show the total elapsed time for both collecting and sensing sensor data to the cloud side in milliseconds.
- The current threshold value of the soil moisture is used to control the water pump at the sensing side. Also, this threshold can be set in the cloud-based windows application.
- Two interactive images are used for illustrating the current status of the farm lighting and the water pump.

[3] XAML is a declarative language that is used to create the GUI of UWP applications.

```
stopWatch.Start();
var events = await eventHubReceiver.ReceiveAsync(int.MaxValue);
stopWatch.Stop();
if (events == null)
{
    if (!anyEventsReceivedBefore)
        stopWatch.Reset();
    continue;
}
anyEventsReceivedBefore = true;
foreach (EventData eventData in events)
{
    string data = Encoding.UTF8.GetString(eventData.Body.Array);
    var enqueuedTime = eventData.SystemProperties.EnqueuedTimeUtc.ToLocalTime();
    var connectionDeviceId = eventData.SystemProperties["iothub-connection-device-id"].ToString();
    messagesCount++;
    if (devices.Contains(connectionDeviceId))
    {
        txtMsgReceived.Text += $"{messagesCount} - {enqueuedTime} , Partition ({partition}), Sensors Data : [ {data} ]";

        if (eventData.Properties.Count > 0)
        {
            txtMsgReceived.Text += "   Alerts: ";

            foreach (var property in eventData.Properties)
            {
                if (property.Key == "SentMsgCount")
                    sentMsgCount = int.Parse(property.Value.ToString());
                if (property.Key == "NoOfDevices")
                    sentMsgCount = sentMsgCount * int.Parse(property.Value.ToString());
                if (property.Key == "DrynessAlert")
                {
                    txtMsgReceived.Text += $"'{property.Key}' : '{property.Value}'\n\r";
                    string currentDrynessAlert = property.Value.ToString().ToLower();

                    if (previousDrynessAlert != currentDrynessAlert)
                    {
                        previousDrynessAlert = currentDrynessAlert;

                        if (property.Value.ToString().ToLower() == true.ToString().ToLower())
                            await AzureIoTHub.InvokeDirectMethod(AzureIoTHub.DirectMethodAction.StartWaterPump);
                        else
                            await AzureIoTHub.InvokeDirectMethod(AzureIoTHub.DirectMethodAction.StopWaterPump);
                    }
                }
            }
        }
    }
}
```

FIGURE 5 Code Snippet for Receiving and Parsing the Telemetry Messages in the Windows Application

Each telemetry message sent from the cloud side contains three sensed readings (i.e., the current moisture, temperature, and humidity reading), in addition to one extra value for the dryness alert property. If a dryness alert message is received, the application turns on the water pump immediately by invoking a cloud pre-registered direct method call on the IoT gateway. The water pump keeps irrigating the field until the sensed moisture level exceeds the moisture threshold.

We also developed an Android mobile app for enabling the user to manually control both the irrigation pump and the farm lighting using (shown in Figure 7). The app includes a voice assistant which uses voice recognition and speech synthesis to translate the voice commands of the user to actions. For example, if the user says, "start irrigation," then the water pump will be turned on. Additionally, the user can turn on/off the farm lighting and the water pump by taping the corresponding icons on the app.

The mobile app was developed using the Xamarin platform[18], which is a development platform for creating native mobile apps across different platforms (e.g. Android, iOS, etc.). Particularly, we used the *Xamarin.Android* library which exposes the complete Android SDK for .NET developers to build fully native Android apps using C# in MS Visual Studio Development Environment.

Figure 8 shows a code snippet for how to control the end-devices using voice recognition in the Android app. We first convert the recorded speech to text. Then, we match the generated command to one of the existing direct method actions. If a matching action is found, we call the appropriate direct method, which in turns sends a direct method call to the device to take action.

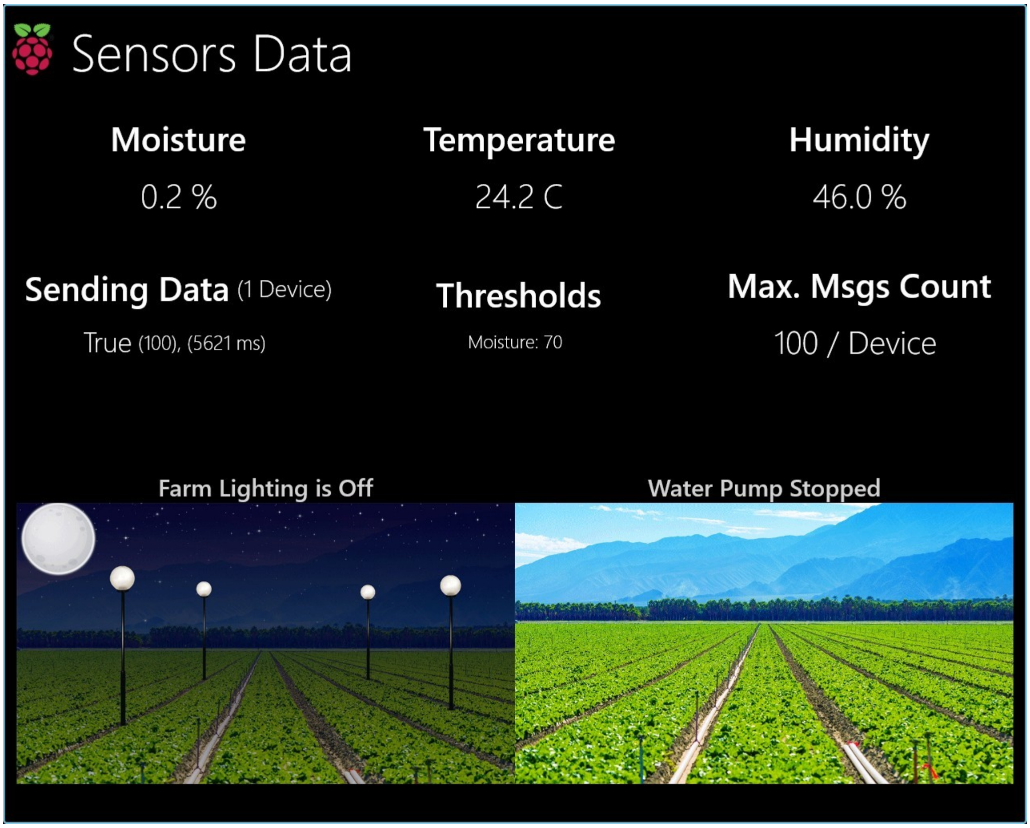

FIGURE 6 Screenshot of the UWP Application Running on the Raspberry Pi

5 | EVALUATION

We experimentally evaluated our prototype regarding performance and scalability. We installed instrumentation in both the cloud-based application and the UWP application running on the Raspberry Pi to measure the processor time that was taken to perform various tasks. Instrumentation was also added to the sensing side to measure the processor time of sensing data. Each experiment presented in this section is carried out for ten trials, then we took the average of these trials' results.

5.1 | The effect of changing the number messages on the response time

We ran a set of experiments to determine the impact of changing the number of messages exchanged between the sensing and cloud sides on the processing time of these messages at the UWP application running on the Raspberry Pi. This application was developed to asynchronously send one sensor feed to the IoT Event Hub per second. In these experiments, we used one IoT end-device to send/receive all messages to/from the cloud side.

Figure 9 shows the results of these experiments. As shown in the figure, as the number of messages increases, the total processing time increases until reaching 500 messages. At this point, we could not observe noticeable significant differences in the response time. The overall average latency was measured to be 0.027seconds per message, which is considered an acceptable latency that makes our system a near-real-time irrigation system. This means when the dryness alert is fired, the irrigation process can start within one second considering the network delay as well. Furthermore, we assumed that a sensor feed is collected every second, and consequently, a message is sent to the cloud side every second, which may be not very reasonable in reality as farmers need to check the moisture level in the soil every couple of hours, for instance.

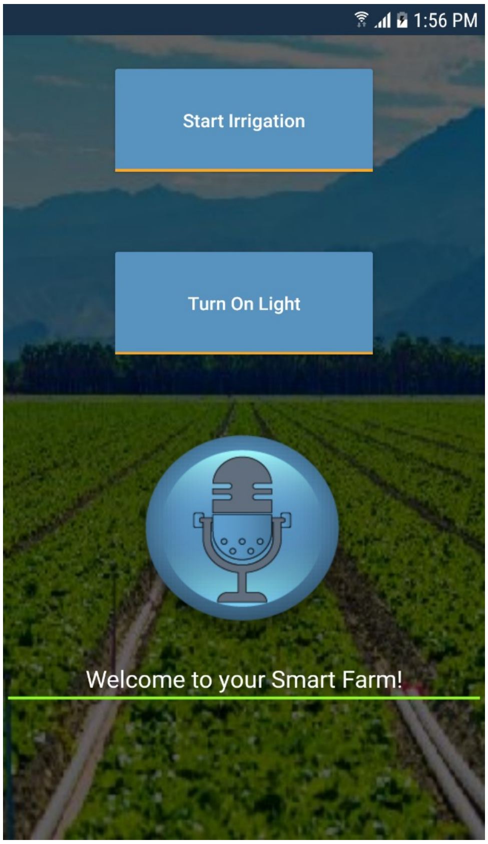

FIGURE 7 The Android Mobile App

5.2 | The effect of changing the number of IoT end-devices on the response time

We ran another set of experiments to determine the impact of changing the number of IoT end-devices on the response time of the system. In the first experiment, we used one IoT end-device to perform all computations. Then, we gradually increased the number of devices in the following experiments. In these experiments, we used the physical pump in addition to other four virtual devices, which are all connected to the IoT hub. We instructed each device to send 100 feeds per second to the cloud side during the time of the experiment.

Figure 10 shows the results of these experiments. As shown in the figure, as the number of devices increases, the latency time slightly increases. This demonstrates that our system is scalable and can support a high number of IoT-end devices without significantly affecting the responsiveness of the system.

5.3 | The effect of changing the number of IoT end-devices on the processing time

Finally, we ran a set of experiments to determine the impact of changing the number of IoT end-devices on the on-going per-event processing time. The main objective of these experiments is to assess the scalability of the system to accommodate an enormous number of sensing devices. The on-going processing time measured was per sensor feed: every time a piece of raw data was received from a sensor, its average the total processing cost amounted to that per-event processing time.

In the first experiment, we used one virtual end-device to perform all computations. Then, we gradually doubled the number of virtual devices in the following experiments. Figure 11 shows the results of these experiments. As shown in the figure, as the number of devices increases, the average processing time per event slightly increases. These experiments show that our system can include a large number of IoT-end devices without having a significant effect on the on-going processing time of sensor feeds.

```
protected override async void OnActivityResult(int requestCode, Result resultVal, Intent data)
{
    base.OnActivityResult(requestCode, resultVal, data);

    if (requestCode == VOICE)
    {
        if (resultVal == Result.Ok)
        {
            var matches = data.GetStringArrayListExtra(RecognizerIntent.ExtraResults);
            if (matches.Count != 0)
            {
                string saidText = matches[0];

                // limit the output to 300 characters
                if (saidText.Length > 300)
                    saidText = saidText.Substring(0, 300).ToUpperInvariant();
                txtInputWhatJustSaid.Text = saidText = CapitilizeFirstLetterEachWord(saidText);
                if (saidText == Application.Context.GetString(Resource.String.farm_water_pump_on))
                {
                    await AzureIoTHub.InvokeDirectMethod(AzureIoTHub.DirectMethodAction.StartWaterPump);
                }
                else if (saidText == Application.Context.GetString(Resource.String.farm_water_pump_off))
                {
                    await AzureIoTHub.InvokeDirectMethod(AzureIoTHub.DirectMethodAction.StopWaterPump);
                }
                else if (saidText == Application.Context.GetString(Resource.String.farm_light_turn_on))
                {
                    await AzureIoTHub.InvokeDirectMethod(AzureIoTHub.DirectMethodAction.TurnOnFarmLight);
                }
                else if (saidText == Application.Context.GetString(Resource.String.farm_light_turn_off))
                {
                    await AzureIoTHub.InvokeDirectMethod(AzureIoTHub.DirectMethodAction.TurnOffFarmLight);
                }

            }
            else
                txtInputWhatJustSaid.Text = "No speech was recognised";

        }
    }
}
```

FIGURE 8 Code Snippet for Controlling the End-devices using Voice Recognition in the Mobile App

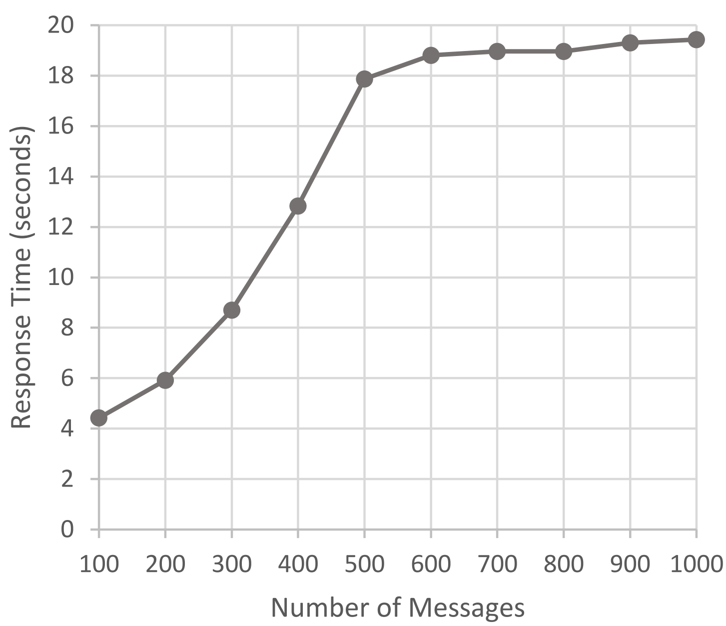

FIGURE 9 The Effect of Changing the Number Messages on the Response Time

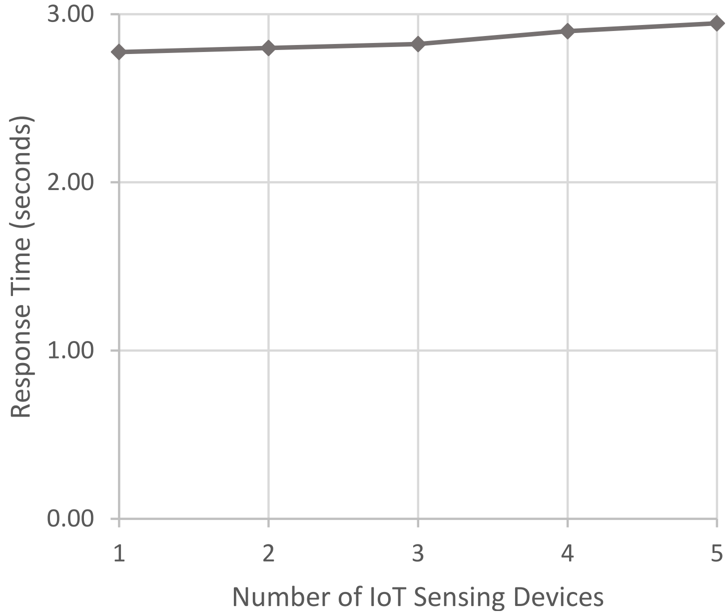

FIGURE 10 The Effect of Changing the Number IoT End-devices on the Response Time

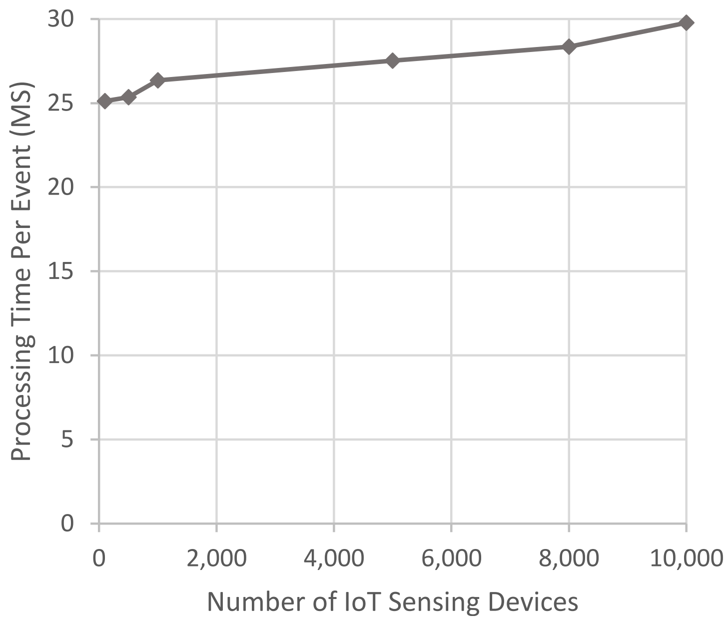

FIGURE 11 The Effect of Changing the Number IoT End-devices on the Ongoing Per-event Processing Time

6 | CONCLUSIONS

This paper presented a distributed IoT system which can better inform and engage farmers with the automated irrigation process in agriculture fields. The developed system would create a better opportunity for farmers to irrigate more efficiently, remotely, as well as reducing the field's overwatering based on actual soil watering needs. We developed three different types of applications as part of the IoT system executing on sensing IoT devices, Azure cloud platform, as well as on users' mobile devices. These applications give the ability to users to set various irrigation parameters such as the thresholds for the moisture, temperature and humidity sensors, which makes the system widely useable for a different type of crops and soils considering they have their suitable soil moisture threshold values. We also carried out several sets of experiments for evaluating the performance and scalability of our system, paying particular attention to establishing the relationship between the number of IoT end-devices connect to the IoT hub and the response time of the system. The results showed that the response time depends on various granularity characteristics of the systems, most notably the number of messages exchanged between the IoT end-devices and the IoT Event Hub. Also, the experimental evaluation showed that the system is highly responsive and scalable despite the number of messages exchanged as well as the number of contributing IoT end-devices.

We expect that this research would increase the open source knowledge base in the area of IoT-based distributed and mobile systems by publishing the source codes to the public domain[4].

In on-going work, we are looking into opportunities for generalizing our approach to be used for other smart farming practices such as automatic seeding, fire detection, lighting, and climate control. This would lessen the need for human interaction and ultimately improve agriculture productivity. We also plan to check the forecasted rainfall in the next few hours before starting the irrigation process. Also, we will test the developed prototype in the Research Farm at Prairie View A&M University, as a pilot site. Furthermore, we are also looking at the opportunity of using deep learning to predict more accurate plant watering requirements. Finally, experiments with more massive datasets are needed to study the robustness of our solution further.

References

1. EPA: Statistics and Facts. https://www.epa.gov/watersense/statistics-and-facts; accessed November 03, 2020.
2. The rise of big data on cloud computing: Review and open research issues. *Information Systems* 2015; 47: 98–115.
3. Remote sensing for irrigated agriculture: examples from research and possible applications. *Agricultural Water Management* 2000; 46(2): 137– 155.
4. Tzounis A, Katsoulas N, Bartzanas T, Kittas C. Internet of Things in agriculture, recent advances and future challenges. *Biosystems Engineering* 2017; 164: 31–48. doi: https://doi.org/10.1016/j.biosystemseng.2017.09.007
5. Nawandar NK, Satpute VR. IoT based low cost and intelligent module for smart irrigation system. *Computers and Electronics in Agriculture* 2019; 162: 979–990. doi: https://doi.org/10.1016/j.compag.2019.05.027
6. Moamen AA, Jamali N. Coordinating Crowd-Sourced Services. In: ; 2014; Alaska, USA: 92–99.
7. Moamen AA, Jamali N. An Actor-Based Middleware for Crowd-Sourced Services. *EAI Transactions on Mobile Communications and Applications* 2017(vol 17): 1–15.
8. Azure IoT Hub. https://azure.microsoft.com/en-us/services/iot-hub/; accessed November 03, 2020.
9. Aleotti J, Amoretti M, Nicoli A, Caselli S. A Smart Precision-Agriculture Platform for Linear Irrigation Systems. In: ; 2018: 1-6.
10. Rajendrakumar S, Parvati VK, Rajashekarappa , D PB. Automation of Irrigation System Through Embedded Computing Technology. In: ICCSP '19. ; 2019: 289–293.
11. Kwok J, Sun Y. A Smart IoT-Based Irrigation System with Automated Plant Recognition Using Deep Learning. In: ICCMS 2018. ; 2018: 87–91.
12. Rao RN, Sridhar B. IoT based smart crop-field monitoring and automation irrigation system. In: ; 2018: 478–483.
13. Goap A, Sharma D, Shukla A, Krishna CR. An IoT based smart irrigation management system using Machine learning and open source technologies. *Computers and Electronics in Agriculture* 2018; 155: 41–49. doi: https://doi.org/10.1016/j.compag.2018.09.040
14. Muangprathub J, Boonnam N, Kajornkasirat S, Lekbangpong N, Wanichsombat A, Nillaor P. IoT and agriculture data analysis for smart farm. *Computers and Electronics in Agriculture* 2019; 156: 467–474. doi: https://doi.org/10.1016/j.compag.2018.12.011

[4] Available online: https://github.com/ahmed-pvamu/Smart-IoT-Irrigation-System

15. Saraf SB, Gawali DH. IoT based smart irrigation monitoring and controlling system. In: ; 2017: 815–819.

16. Sales N, Remédios O, Arsenio A. Wireless sensor and actuator system for smart irrigation on the cloud. In: ; 2015: 693–698.

17. Azure IoT Event Hub. https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-about; accessed November 03, 2020.

18. Visual Studio Tools for Xamarin. https://visualstudio.microsoft.com/xamarin/; accessed November 03, 2020.