Electronics and Electrical Engineering

https://www.aimspress.com/journal/ElectrEng

AIMS Electronics and Electrical Engineering, 4(1): 57–86.
DOI: 10.3934/ElectrEng.2020.1.57
Received: 07 October 2019
Accepted: 27 November 2019
Published: 12 December 2019

*Research article*

# Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol

**Lawrence O. Aghenta*and M. Tariq Iqbal***

Department of Electrical and Computer Engineering, Faculty of Engineering and Applied Science, Memorial University of Newfoundland (MUN), St. John's, NL A1B 3X5, Canada

* **Correspondence:** Email: loaghenta@mun.ca, tariq@mun.ca; Tel: +1 709 864 8934.

**Abstract:** Distributed assets, such as hybrid power system components, require reliable, timely, and secure coordinated data monitoring and control systems. Supervisory Control and Data Acquisition (SCADA) is a technology for the coordinated monitoring and control of such assets. However, SCADA system designs and implementations have largely been proprietary, mostly pricey and therefore economically unjustifiable for smaller applications. With proprietary SCADA systems, there is also the problem of interoperability with the existing components such as power electronic converters, energy storage systems, and communication systems since these components are usually from multiple vendors. Therefore, an open source SCADA system represents the most flexible and most cost-effective SCADA option for such assets. In this paper, we present the design and implementation of a low-cost, open source SCADA system based on the most recent SCADA architecture, the Internet of Things (IoT). The proposed SCADA system consists of current and voltage sensors for data collection, an ESP32 micro-controller with organic light-emitting diode (OLED) display, for receiving and processing the sensor data, and ThingsBoard IoT server for historic data storage and human machine interactions. For the sensor data transfer from the ESP32 to the ThingsBoard IoT server, Message Queuing Telemetry Transport (MQTT) protocol is implemented for data transfer over a local Wi-Fi connection with the MQTT Client configured on the ESP32, and the ThingsBoard server node serving as the MQTT Broker. The ThingsBoard IoT server is locally installed with PostgreSQL database on a Raspberry Pi single-board computer and hosted locally on MUN Network for data integrity and security. To test the performance of the developed open source SCADA system solution, it was setup to acquire and process the current, voltage and power of a standalone solar photovoltaic system for remote monitoring and supervisory control. The overall system design procedures and testing, as well as the created dashboards and alarms on the ThingsBoard IoT server platform are presented in the paper.

**Keywords:** open source; SCADA; ThingsBoard; Internet of Things; ESP32 with OLED; Raspberry Pi; MQTT; automation; instrumentation and control

## 1. Introduction

Energy shortage and Global Warming are some of the major challenges facing the world today, especially with the recent rapid industrial development across the globe. As such, world leaders in collaboration with energy experts continue to search for alternative sources of energy, such as clean and renewable energy, to meet the growing global demand for energy and to save the environment from further degradation, caused by the use of the conventional sources of energy such as fossil fuels over the years. As energy experts continue to capture clean and renewable energy sources for the benefit of mankind, these sources are continuously being injected into today's power systems. These clean and renewable sources are incorporated with the conventional energy generation systems to form Hybrid Power Systems (HPS) [1]. However, due to the intermittent nature of these sustainable (renewable) energy sources such as solar and wind as they are hugely affected by the prevalent environmental conditions, energy storage systems are usually required in the resulting hybrid power systems for system, power grid and supply stabilities. Energy storage systems help to mitigate supply output fluctuations, as well as help to ensure frequency control and load balancing, amongst other important functions. These hybrid power systems, which are usually made up of the conventional energy generation sources such as fossil fuels, and one or more renewable generation sources such as wind and solar, together with the energy storage systems, power electronic converters such as inverters, as well as other power system devices such as communication systems needed for their successful operations are usually spread over large geographical areas, sometimes in harsh environments such as offshore and swamps. As a result of this distributed nature, the interconnection of these systems to generate and supply energy presents numerous challenges, such as power quality issues, voltage tolerances, frequency control, grid synchronization and metering, data exchange and communications between components, as well as the safety and security of both assets and personnel [2].

In order to overcome these challenges and to ensure seamless power system operations, diverse sensors, micro-controllers, micro-processors (e.g., programmable logic controllers), actuators, valves, pumps, etc. are usually connected to various points of interest in the entire HPS to acquire important data such as current, voltage, power, etc., and for real-time data monitoring, remote and co-ordinated controls. Supervisory Control and Data Acquisition (SCADA) is the perfect solution for these tasks. Since these hybrid power systems and their associated components are located remotely, a SCADA system is needed for their remote monitoring, coordinated control, and data acquisition from the various sensors, actuators, and other field instrumentation devices connected to the various points of interest. The SCADA system would help in the efficient collection of data from these variously distributed sensors, actuators and controllers, real-time remote control of the system, remote monitoring, and maintenance of the produced current, voltage, and power [3].

SCADA refers to the combination of telemetry and data acquisition. It encompasses the collection of information (data) from distributed process facilities, the transfer of these data to a central location, analysis of these data to know the current states of the distributed process facilities, supervisory control of the process facilities, displaying these data on a number of operator screens or displays (Human Machine Interface), and conveying the necessary control actions back to these distributed process facilities for the local operator's actions [4, 5]. It is a closed loop control system. The major functions of a SCADA system include the following [4]: Data acquisition, Data presentation,

Supervisory control, Networked data communication, Alarm processing, Historic data storage, data trending and reporting, and Remote monitoring. The architectural design of a SCADA system is made up of four basic elements; field instrumentation devices such as sensors which collect data from the distributed process facilities being managed, Remote Terminal Units (RTUs) such as single-board computers (PLCs, micro-controllers, etc.) for acquiring, processing and parsing these sensor data, Master Terminal Units (RTUs) such as IoT servers and platforms for data processing and human machine interactions, and finally SCADA communication channels for connecting the RTUs to the MTUs, and for data transfer [6].

SCADA architectures have evolved over the years, starting from the very first generation SCADA systems in the 70s called Monolithic SCADA, through the second generation SCADA systems called Distributed SCADA (80s and 90s), and the third generation SCADA systems called Networked SCADA (90s and early 2000s), to the most recent SCADA architecture called the Internet of Things (IoT) SCADA architecture (4th generation) [7]. The SCADA system proposed in this work is based on the Internet of Things SCADA architecture. The Internet of Things concept refers to the interconnection of physical objects, embedded electronics, software and sensors, and so on, to enable real-time data exchange and communication between these devices and an operator over a common network or the web [8, 9]. The IoT-based SCADA system incorporates web or cloud services with the conventional SCADA system for a more robust remote monitoring and control [7].

In general, there are two classes of SCADA systems, and they include Proprietary (Commercial), and Open Source SCADA systems [10]. Automation companies like Siemens and Schneider Electric design and develop proprietary SCADA systems such as Simatic WinCC (Siemens), ClearSCADA (Schneider Electric), Ovation SCADA (Emerson), Micro SCADA (Allen Bradley), etc. and they sell these systems as turn-key solutions to the end users while providing regular or scheduled operational and technical supports both remotely and on site (locally). Aside the high initial capital cost of purchasing these SCADA systems, there are usually some additional subscription charges for maintenance and supports which could be billed monthly or quarterly. Thus, these proprietary SCADA solutions are largely expensive and mostly economically unjustifiable for smaller power system applications. In addition to the cost implications of these commercial SCADA system solutions, there is the problem of interoperability with the existing power system infrastructures. This is because the electromechanical components of the hybrid power system, as well as the energy storage systems, power electric converters, and other interconnected devices and the grid integration devices are usually from multiple manufacturers. Seamless integration of a SCADA system into existing infrastructures and communications facilities is of great importance to avoid incurring additional costs due to modifications and redesigns. For these reasons, an open source SCADA system represents the most flexible SCADA solution [10]. Furthermore, in addition to the cost savings of not having to redesign the communication facilities in integrating an open source SCADA system into the existing infrastructures, an open source system allows an end user to ”mix and match” components and choose the most appropriate from various vendors, and as such the end user is not beholden to a single vendor [10]. Therefore, an open source SCADA system represents the most flexible and most cost-effective SCADA system solution.

In this paper, we present the design, development and implementation of a low-cost, open source SCADA system based on the Internet of Things (IoT) SCADA architecture. The SCADA system proposed uses low-cost, low-power, reliable and readily available components to realize the desired

functions of a SCADA system. We show that the proposed SCADA system works well by testing it extensively using a standalone renewable power generation source, solar photovoltaic (PV) system, made up of solar panels and battery energy storage system similar to the arrangements in a small hybrid power system. Specifically, the key contributions of this paper are summarized as follows:

- Implementation of a SCADA data security technique by installing the ThingsBoard IoT Server, which is the main data server, on a local machine (on-premise) and self-hosting it on a private network. This, to the best of our knowledge is novel because in similar open source SCADA solutions reviewed, the authors either used the web-based IoT Platforms or developed web-based servers for data storage and human machine interactions, which could leave the stored data vulnerable to internet attacks just like every other web application.
- Implementation of a local operator real-time data monitoring interface by virtue of the ESP32 OLED display screen configured to display real-time data. This is also new as we have not found such implementations in similar open source SCADA solutions reviewed.
- Finally, after a careful study of the available proprietary (commercial) SCADA solutions, we present a tutorial-like approach for an IoT-based open source SCADA system design and implementation. This will be a valuable guide to anyone planning to carry out such an exercise in the future.

The organization of the remaining part of this paper is as follows. In Section 2, we present the related works, including problem statements, and the proposed SCADA system as a solution to the identified problems. In Section 3, we present brief overviews of the major technologies employed in our proposed solution, including Internet of Things (IoT) and Message Queuing Telemetry Transport (MQTT). The proposed SCADA system architectures are presented in Section 4, the components of the proposed SCADA system and descriptions of each of the components are presented in Section 5, and the implementation methodology used in the data collection, logging and remote monitoring presented in Section 6. In Section 7, we present the hardware implementation, and we present the experimental setup of the proposed SCADA system in Section 8. The testing procedures and the realized results are presented in Section 9, and in Section 10, we present brief discussions of the key features of the developed SCADA system solution, including cost and power consumption analyses. The paper is concluded in Section 11, and future directions of the research presented in Section 12.

## 2. Related works

Numerous attempts have been made to reduce the over-dependence on proprietary (commercial) SCADA systems by designing various forms of low-cost, and open source SCADA systems for different industries and applications. For power system applications for example, a few attempts have been made to develop alternative SCADA systems for critical assets monitoring and remote control. J. Lee et al. [3] have proposed an IoT-based open source SCADA system for the remote monitoring, power control, and distributed data processing of a standalone offshore wave-wind hybrid power generation system based on the IEC61850 standard. In particular, their proposed SCADA system comprised of two control devices; a PLC and an industrial VPN router, and the data acquisition, network communication function, PLC management, and data visualization functions were carried out by the router device. Their proposed solution was tested in a simulation-based testing environment with the power transmission system's operator generating commands. In a similar power

system application, S. A. Alavi et al. [11] presented an IoT-based open source data collection and visualization system. In their proposed system, two ESP-12E network modules were configured to acquire the desired micro-grid data, and to parse the data using MQTT protocol over Wi-Fi in one setup, and MQTT protocol over GPRS in another setup, depending on the desired coverage range. The acquired data were transmitted to the web-based ThingsBoard server where dashboards were created for situational-awareness (SA), data visualization and micro-grid management.

Elsewhere, K. Kao et al. [12] developed an IoT-based SCADA system for inverter monitoring and remote control. In their implementation, they divided their solution into four different levels which they called monitor; server; cloud; and client tiers. The monitor tier comprised of sensors, a Wi-Fi data acquisition device, an inverter, and a wireless router. A PC server served as the server tier, a database served as the cloud tier, and a laptop, a tablet, and a smart phone served as the client tier. The inverter data such as voltage and frequency were transmitted via a Wireless Sensor Network (WSN) to the database in the cloud, and Asynchronous JavaScript and XML (AJAX) and Responsive Web Design (RWD) tools were used to develop the human machine interface (HMI) for inverter data visualization. Remote control of the voltage and frequency of the inverter was also implemented via RS-485 connection and Modbus protocol. In another development, authors in [13] proposed a cloud-based SCADA system by integrating JustIoT framework with the conventional open source SCADA architecture. In their solution, the JustIoT structure was bridged to the conventional SCADA for cloud capabilities using Modbus TCP and OPC client. The JustIoT structure comprised of Raspberry Pi3 and Arduino Uno micro-controllers for data transfer via MQTT to an intelligent server consisting of Firebase cloud system, and a cloud-based real-time database where PC and mobile devices could visualize the stored data. Their designed system was applied in offshore wind power monitoring, and for smart house monitoring.

Although there are some studies on the design of SCADA systems in general for standalone or grid connected hybrid power systems, most IoT-based remote monitoring and control systems, especially using the lightweight data transfer protocol for the Internet of Things called MQTT, have focused on other sectors and their related applications such as smart healthcare applications [14–16], home automation applications [17–21], intelligent voting systems [22], infrastructure and transport applications [23–27], industrial manufacturing environments [28, 29], and so on.

The major issue with most of the reviewed IoT-based open source SCADA systems above is that the solutions are rather cumbersome as they involve a lot of technologies, tools and programming. Although remote monitoring and control is a complex issue, it is important for the solutions to be as simple as possible, especially for an open source system where the deployed system might have to be operated by the facilities' owners independently of the system designer. This is important because the facilities' owners might not have the advanced technological and programming knowledge needed to safely and successfully manage a complex system. In addition to the complex nature of the reviewed solutions, most of the authors either used a web-based IoT platform for data visualization, storage, and other human machine interactions such as the AWS used in [22], and the web-based ThingsBoard platforms implemented in [11, 30], or designed a web platform for data visualization and human machine interactions using multiple web technologies like the AJAX and RWD technologies utilized in [12], and the Google Chrome based application in [17, 24]. The major problem with using web-based platforms for data management in a critical SCADA system is that the stored data are highly susceptible to internet attacks since the web-based platforms require the public internet for data

access just like every other website out there. However, the importance of data security in a SCADA system cannot be overemphasized. This is because attacks on a SCADA system can compromise the critical infrastructures being managed, which could result in devastating economic and operational setbacks. Data integrity in a SCADA system can be ensured by using several techniques, including securing the data communication channel or network such as data encryption, securing the hardware components, or securing the cloud server where the data are stored [4, 7, 31–33].

In this paper, we implement a combination of private network management and private cloud server management strategies to ensure the security of the proposed IoT-based SCADA system. To achieve this, the ThingsBoard IoT Server for data management, storage and human machine interaction is locally hosted on a Raspberry Pi machine, and a private Wi-Fi network is created with a Wi-Fi router while data transfer is made possible using the MQTT data transfer protocol over the Wi-Fi network, such that only the users with the right authorizations can have access to the stored data. This also means that the SCADA system operator has full control of the security of the system, unlike when it is being managed over the public internet. On this private network, the operator can take multiple measures to protect the data in the cloud. Such measures include ensuring access control, whitelists, authentication, authorization, firewalls, regular risk assessment, continuous monitoring and log analysis, updating and patching regularly, etc. In addition to taking these measures to ensure the security of the proposed SCADA system, we implement the lightweight ISO standard data transfer protocol for the Internet of Things, the Message Queueing Telemetry Transport (MQTT) protocol, for the sensor data transfer from the MQTT client (ESP32 device) to the Raspberry Pi-installed ThingsBoard IoT Server platform which serves as the MQTT broker.

In our proposed SCADA system design solution, very low-cost, low-power, and completely open source components are used as the elements of the SCADA system. A locally installed ThingsBoard IoT server on a Raspberry Pi machine serves as the Master Terminal Unit (MTU) for data storage, data visualization and human machine interactions; ESP32 micro-controller, with OLED display, serves as the Remote Terminal Unit (RTU) for receiving and parsing the sensor data from the Field Instrumentation Devices (Current and Voltage Sensors); and a local Wi-Fi network created with a Wi-Fi router serves as the SCADA Communication Channel between the MTU and the RTU, and over which MQTT data transfer protocol is used for data transfer from the RTU (client) to the MTU (broker). The entire system forms a kind of secure industrial network as implemented by the commercial SCADA manufacturers in various industrial domains all over the world [34]. To the best of our knowledge, we have not found a single literature where a locally installed ThingsBoard IoT Server has been used as the MTU in an IoT-based SCADA system design. Furthermore, with the organic light-emitting diode (OLED) display of the ESP32 device (RTU) used in our design, we ensure that a local operator is able to visualize the current state of the process plant being managed by seeing the data values on the OLED screen, in addition to receiving updates from the remote SCADA operator at the server side. This is also an additional SCADA feature considered in this work.

## 3. Overview of technologies

In this section, we present a brief description of each of the major technologies employed in the design of our proposed open source SCADA system solution as they relate to the subject matter at hand. These technologies include Internet of Things (IoT), a technology upon which the SCADA

architecture considered is built; and Message Queuing Telemetry Transport (MQTT), a lightweight data transfer protocol for the IoT domain.

### *3.1. Internet of Things (IoT)*

The Internet of Things (IoT) is a technology that enables the interconnection of physical devices such as buildings, vehicles, etc. with embedded electronics, sensors, softwares, and network connectivity such that the devices can collect and exchange real-time data between themselves and an operator over a common platform, the web or network [8, 9, 35]. The IoT concept is made possible by the exploitation of existing technologies and concepts such as pervasive and ubiquitous computing, embedded devices, communication technologies, internet protocols and applications, and sensor networks which help in the transformation of these physical devices from their traditional forms into smart devices [8]. In the last few years, the IoT concept has been exploited in homes, schools, businesses and industrial domains to bring about smart cities, smart homes, smart healthcare systems, smart energy management systems, smart transportation, smart manufacturing systems, industrial automation systems, smart emergency response systems, and so on [8, 35]. Even though the IoT technology has been around for sometime now, the subject matter experts have not fully agreed on a standard IoT architecture. Different researchers have proposed various architectures [8, 9]. However, the most common architectures are the three-layer and the five-layer architectures. These are extensively discussed in [8, 9]. Whichever IoT architecture is implemented, a reliable communication is necessary in an IoT-based system as the IoT devices are usually dispersed over large geographical areas. IoT technologies make use of the four layers of the general TCP/IP model. The relationship between the TCP/IP model and the IoT protocols is shown in Figure 1 [35, 36].

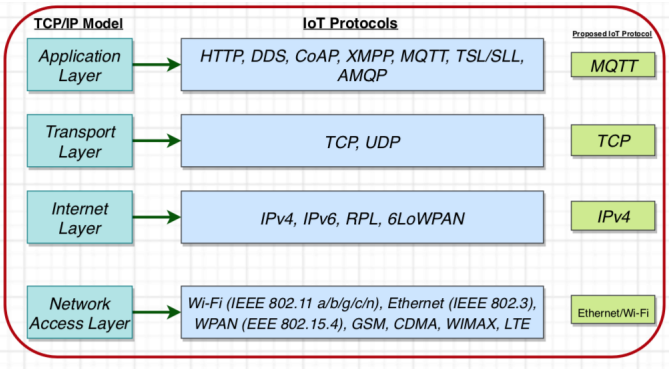


**Figure 1.** TCP/IP model vs Internet of Things (IoT) protocols.

The SCADA system proposed in this work is based on the IoT-SCADA architecture in which IoT features are incorporated into the conventional SCADA system for more robust data acquisition, remote monitoring and supervisory control. This means that IoT protocols are involved in the IoT-based SCADA system design. Research has shown that the most critical design alternatives for developing IoT-based real-time applications such as SCADA, are communication protocols, message encoding format, and the web or IoT platform [35, 36]. Therefore, in each of the IoT protocol layers shown in Figure 1, it is critical to pick the right protocol for a particular IoT-based application. For instance, in the Application Layer, one has to make the hard choice between HTTP, MQTT, CoAP, and so on, as well as pick the right protocol in the Transport, Internet, and Network Access Layers.

The properties, advantages and disadvantages of the different Application Layer IoT protocols such as MQTT, CoAP, HTTP, and AMQP are extensively discussed in [8, 35, 37]. In this work, after so much research, testing and consultations, we have chosen to go with MQTT, TCP, IPv4, and IEEE 802.11 (Wi-Fi)/Ethernet in the respective Layers. In our proposed system, MQTT data transfer protocol is implemented over TCP/IP Wireless connectivity. ThingsBoard has also been chosen as the preferred IoT platform after researching the alternatives [38], and for security reasons, the ThingsBoard IoT server platform is locally installed and hosted on own private machine (Raspberry Pi), and own network (Memorial University (MUN) Network) rather than using the ThingsBoard web platform as is commonly implemented in literatures. One of the reasons for choosing ThingsBoard is that it supports MQTT protocol over wireless connectivity. When implemented, the ThingsBoard server API serves as MQTT Broker [39]. MQTT is discussed in the next section.

### *3.2. Message Queuing Telemetry Transport (MQTT) protocol*

Message Queuing Telemetry Transport (MQTT) is a lightweight machine-to-machine data communication protocol [8]. The MQTT, with a 2 byte fixed header, is especially suited for IoT applications as it supports applications with limited resources such as low bandwidths, low computational power, low memory, battery, etc. which is typical in an IoT domain [17, 22, 24]. MQTT runs on TCP/IP connection, and can be implemented on various networks such as Wired (Ethernet) and Wireless Local Area Networks [26, 40]. Originally invented in 1999 by Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom (now Eurotech), MQTT is now recognized as an open standard by the Organization for the Advancement of Structured Information Standards (OASIS) [27].

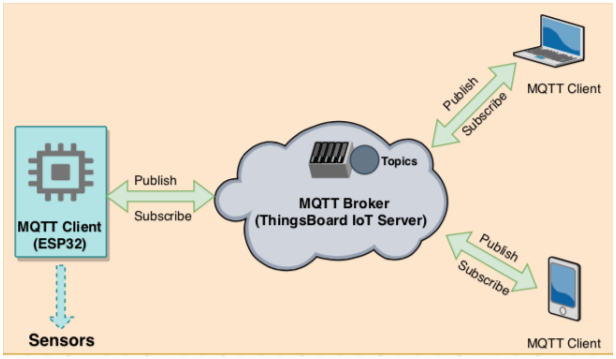


**Figure 2.** MQTT architecture.

Figure 2 shows the general architecture of the MQTT protocol. Essentially, MQTT uses a Publish-Subscribe messaging mechanism, and it is made up of a Broker (Server), and Clients. The MQTT Broker is usually a server running in the cloud or on the internet, and is responsible for storing the published data based on different Topics, and releasing the message (data) to the rightful Subscribers. The MQTT Client is any piece of hardware, software, or a combination of both hardware and software, which connects to the Broker for the purpose of exchanging data. When a connected device or Client downloads data from the Broker (server), the process is called Subscribing, and that particular Client is known as a Subscriber. On the other hand, when a connected device (Client) sends data to a Broker (server), the Client is referred to as the Publisher, and the process is called Publishing. Hence, the term, "Publish-Subscribe" mechanism upon which the MQTT protocol is based. It is worth noting

that a Broker can serve several Subscribers and Publishers, depending on the capability of the machine running the Broker. A Client can also act as both a Publisher and a Subscriber. When a Client publishes data, it assigns a specific Topic to the data, with each Topic separated by a forward slash. For each Topic, there is a Topic name, and Topic level associated with it, and wildcard characters are used to match multiple levels to a Topic [8, 17, 26].

MQTT protocol uses three levels of Quality of Service (QoS) for message delivery: QoS 0, QoS 1, and QoS 2. Thus, MQTT protocol is reliable and suitable for IoT applications where there are usually limited resources such as low bandwidth, low memory, low power, etc [8, 17, 22, 26]. This is why MQTT is preferred to other data transfer protocols like CoAP, HTTP, REST API, etc [8, 35]. MQTT also runs on TCP/IP connection unlike CoAP which runs on UDP [8, 35]. The detailed advantages of MQTT over other data transport protocols are presented in literatures [8, 35], and [37]. Because of these valuable features of the MQTT protocol, numerous applications already use it. For example, Facebook messaging, smart home monitoring, healthcare, transport, energy metering, parking, industrial robots, manufacturing, and intelligent monitoring systems have been implemented with MQTT protocols over various TCP/IP connections [14–29]. In this project, MQTT protocol is implemented for PV data transfer from the MQTT Client (ESP32 micro-controller) to the MQTT Broker (ThingsBoard IoT server), while personal computers and mobile devices can subscribe to visualize the published data on the server. ThingsBoard server node acts as an MQTT Broker, and it supports QoS levels 0 (at most once) and 1 (at least once), and a set of predefined Topics [39]. This ThingsBoard IoT server together with PostgresSQL Database is built on a Raspberry Pi 2 micro-controller, and the MQTT Client Library (Arduino PubSubClient) is implemented with Arduino IDE Software on an ESP32 OLED micro-controller board to collect the sensor data and publish them to the ThingsBoard IoT server (MQTT Broker).

## 4. Proposed SCADA system architecture

In this section, we describe the hardware and network structures of the proposed IoT-based open source SCADA system solution. In configuration A (Figure 3), the Raspberry Pi 2 micro-controller hosting the ThingsBoard IoT server (MQTT Broker) where the received PV data are processed and stored is connected through an Ethernet cable to MUN network. As a result of this connection, authorized users on MUN network can access the stored PV data and visualize the created dashboards and alarms on the ThingsBoard IoT server. In addition, authorized personnel can also have access to the stored data via the internet by using their private home or office network as long as the ThingsBoard IoT server port is opened only on the MUN network. Although this configuration presents a great deal of flexibility as it provides many options for the stored data access, it poses security risks to the stored data since external internet users can access the stored data remotely. In configuration B (Figure 4), the public internet is not used, and the Wi-Fi Router is used to create a form of industrial network such that only the authorized users nearby can access the stored PV data on the ThingsBoard IoT server platform. This is done by connecting the Raspberry Pi machine hosting the ThingsBoard IoT server to one of the LAN ports of the Wi-Fi Router. In this configuration, only the connected and authorized users on the local Wi-Fi network created with the Router can access the data on the ThingsBoard IoT server by pointing to the IP address of the Raspberry Pi on their browsers. This configuration also ensures that even the users on the general MUN network cannot

access the stored data in the cloud, thereby guaranteeing the security of the stored data. Firewall, and authentications are setup on the Router to guarantee the security of the system. In both configurations, the MQTT Client (ESP32 OLED device) processes and publishes the sensor data to the MQTT Broker (ThingsBoard server) using MQTT protocol on the TCP/IP Wi-Fi connection established with the Router.

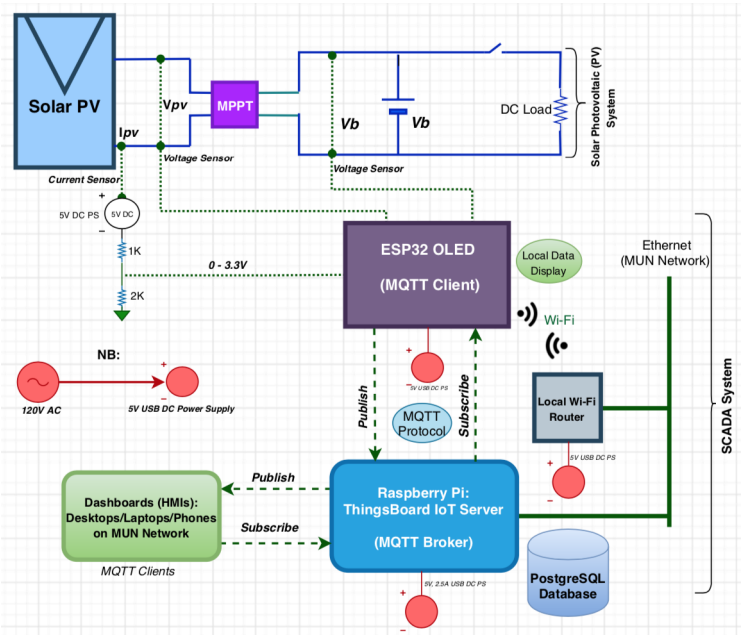


**Figure 3.** The proposed SCADA system configuration A.

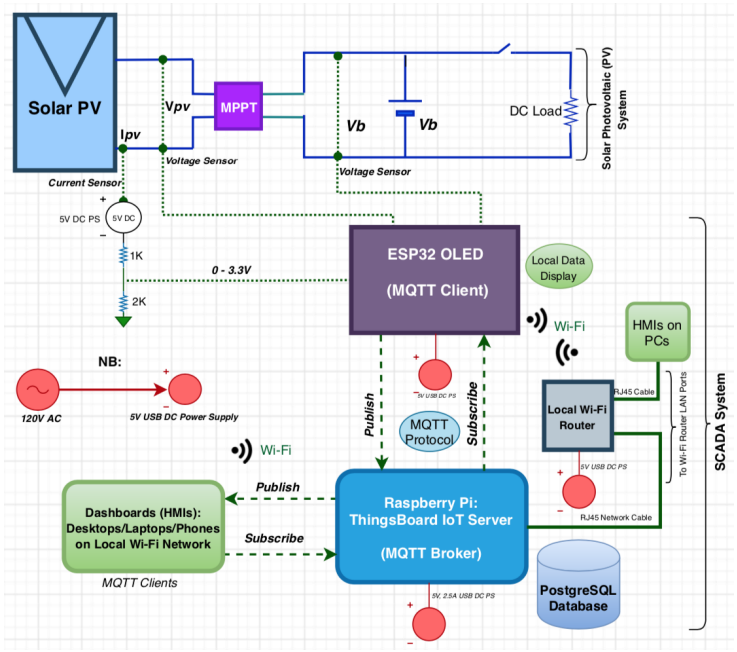


**Figure 4.** The proposed SCADA system configuration B.

## 5. Proposed SCADA system components

Here, we present a brief description of each of the low-cost hardware and software components used in the realization of the proposed open source SCADA system design. These components include the Hall Effect Current and Voltage Sensors which serve as the field instrumentation devices to acquire the PV system data, the versatile ESP32 micro-controller, with OLED display, which is the remote terminal unit, and is configured as the MQTT Client to process and publish the sensor data using MQTT protocol, a Raspberry Pi2 single-board computer upon which the ThingsBoard IoT server, which is the master terminal unit, and is configured as the MQTT Broker, is built for human machine interactions, data storage, dashboards, alarms, data publishing and subscription, and finally, a Wi-Fi Router for creating the TCP/IP Wi-Fi connection for the MQTT protocol implementation.

### *5.1. Sensors (field instrumentation devices)*

Three sensors are used in this work, including two MH Electronic Voltage Sensor modules, and one ACS 712 Hall Effect Current Sensor. These are described in the next sub-sections.

#### 5.1.1. MH Electronic voltage sensors

This low-cost analog voltage sensor uses the concept of voltage divider to measure voltage with its in-built series connection of a 7.5 K resistor and a 30 K resistor. Its operating voltage range is 3.3 V to 5.0 V, and it is capable of detecting supply voltages in the range of 0.025 V to 25 V using a 12-bit ADC [41]. The two voltage sensors used in this project are connected as follows: One of the sensors is connected in parallel across the solar PV system to measure the PV Voltage, while the second sensor is connected in parallel across the lead acid storage battery system (after the MPTT module) to measure the storage battery voltage. The outputs of the sensors are connected to the ESP32 OLED micro-controller as follows: For the first voltage sensor, PIN S is connected to Analog PIN 34 on the ESP32, and PIN – is connected to a GND pin on the ESP32 while its GND and VCC pins are connected in parallel across the PV panel (PIN + of the sensor is not used). For the second voltage sensor, PIN S is connected to Analog PIN 35 on the ESP32, and PIN – is connected to a GND pin on the ESP32 while its GND and VCC pins are connected (after the MPPT module) in parallel across the storage battery (PIN + of the sensor is not used) [41].

#### 5.1.2. ACS 712 Hall Effect current sensor

This low-cost, fully integrated current sensor is manufactured and supplied by Allegro MicroSystems, LLC. The sensor is based on the principle of Hall Effect. The 30 A DC module used in this work has 66 to 185 mV/A output sensitivity, and it operates on a 5 V single supply voltage. When this 5 V supply voltage is applied, the current flowing through the copper conduction path generates a magnetic field which the Hall IC then converts to a proportional output voltage. Using the sensor sensitivity, the signal voltage, and the ADC resolution of the ESP32 micro-controller, the equivalent output current is calculated from this output voltage using the Arduino IDE software. Using this system, 0 A corresponds to 2.5 V, and 30 A corresponds to 5 V on the 30 A DC module used in this setup. However, because the signal voltage of the current sensor is 5 V, it is not suitable for direct connection to the ADC pins of the ESP32 micro-controller as the ADC pins operate

between 0 V to 3.3 V. Therefore, to ensure the accuracy of the measured values, a pull-down or step-down resistors arrangement is used to match the 5 V signal requirement of the current sensor to the 3.3 V signal capability of the ESP32 ADC pins. This is done using a voltage divider equation. With the help of this step-down resistors connection, the current sensor is connected to the ESP32 micro-controller as follows: Its OUT pin is connected through the pull-down resistors to the Analog pin 32 on the ESP32, its GND pin is connected to a GND pin on the ESP32, and its VCC pin is powered with a 5 V supply. In order for the sensor to measure the current flowing through the PV panel, its input pins are connected in series to the PV panel [41].

### *5.2. TTGO ESP32 LoRa32 OLED micro-controller (RTU)*

The TTGO ESP32 LoRa32 micro-controller board with 0.96 inch organic light-emitting diode (OLED) display used in this work has the following key specifications [42]:

- IEEE 802.11 b/g/n Standards HT40 Wi-Fi Transceiver.
- Dual Core Processor, clocked at 240 MHZ; 520 KB Static RAM (SRAM).
- 4MB on-board Flash, and Wi-Fi and Bluetooth Antenna and BLE low-power Bluetooth support.
- 18 ADC pins and over 30 GPIO pins (I/O Pins); 0.96 inch white OLED display.
- 5 V single power supply, and support for a single-cell lithium-polymer (LiPo) battery.
- Operating ADC signal voltage range of 1.8–3.7 V; Built-in heat sink.

With these specifications, especially with the 520 KB SRAM and 4 MB Flash, the board supports debugging, as well as programming the firmware after the development of an application. Also, with the Wi-Fi support, the board can implement HTTP and MQTT. In this work, the ESP32 micro-controller is programmed as an MQTT Client using the Arduino IDE software and the MQTT Client Library called PubSubClient. The program is such that the board acquires the measured real-time PV voltage and current values, battery voltage values, and the calculated PV power values from the sensors, displays the values on both the Arduino IDE Serial Monitor and its OLED display screen, and continuously publishes the real-time data to the MQTT Broker, the ThingsBoard local server IoT platform, using MQTT protocol over the TCP/IP Wi-Fi connectivity. The ESP32 board represents the Remote Terminal Unit (RTU) in the proposed IoT-based SCADA system, and the MQTT protocol implemented on the board helps in realizing the basic functions of the RTU: to acquire and process the sensor data, and parse them to the Master Terminal Unit, ThingsBoard Server in this case. The OLED display screen of the board provides the extra SCADA monitoring function of giving the local plant operator an interface to view the most recent data of the facility being managed. Although the local plant operator is unable to see the data trends on the OLED display as is on the HMI on the Master Terminal Unit (SCADA server), being able to see the most recent data could be critical in the event that the remote SCADA operator is unavailable to provide supervisory control actions. This extra SCADA feature implemented in this work is one of the contributions of this work as we have not found any IoT-based open source SCADA system design solution where such extra plant monitoring feature has been implemented.

### *5.3. Raspberry Pi single-board computer*

The Raspberry Pi 2 model B device used in this work is a low-cost, single-board computer featuring the BCM2836 quad core ARMv7 processor. The Raspberry Pi machine primarily runs on

Raspbian operating system (OS) which is a Debian-based Linux distribution specifically for the Raspberry Pi [43]. Its most important hardware specifications include [43]: 32-bit 900 MHz quad-core ARM Cortex-A7 CPU, 1 GB RAM, 100 Base Ethernet support, 40 GPIO pins, 4 USB ports, Full HDMI port, Micro SD card slot, and so on. In the proposed SCADA system design, the ThingsBoard IoT server which is the MQTT Broker, as well as the Master Terminal Unit for human machine interactions and data processing, is installed on the Raspberry Pi computer. This is done by building the ThingsBoard server alongside the required third-party components and services on the Raspberry Pi operating system. To realize this, the Raspbian Buster OS was installed on a micro-SD card and inserted into the Raspberry Pi's micro-SD card slot. Following the available ThingsBoard installation documentation [44], and the open source ThingsBoard server repository on GitHub, several installation codes were run on the Raspberry Pi Terminal to download and install the ThingsBoard version 2.4 server and its services on the Raspberry Pi computer. The open source PostgreSQL Database was also installed alongside the ThingsBoard server to store the acquired data. After the installation and configuration of the ThingsBoard server and the PostgreSQL Database on the Raspberry Pi, the Raspberry Pi was then connected to the other components of the proposed SCADA system in two different configurations as described earlier, and each configuration was tested.

#### 5.3.1. Wi-Fi Router (TCP/IP Wi-Fi connection)

The D-Link Router (DI-524 Airplus G) is used to create the TCP/IP Wireless network connectivity over which the MQTT protocol is implemented for data transfer from the MQTT Client (ESP32) to the MQTT Broker (ThingsBoard IoT server). Because the ESP32 micro-controller used in this work supports TCP/IP, IEEE 802.11b/g/n Wi-Fi standards, the router is configured to set up the needed local Wi-Fi network connectivity, and the implemented MQTT protocol on the ESP32 (MQTT Client) is used to publish the acquired sensor data over this TCP/IP Wireless connectivity to the ThingsBoard IoT server platform (MQTT Broker). For security and access control purposes, connection to the Wi-Fi network is restricted using the Wi-Fi network name (SSID) and the assigned password, as well as other implemented security measures on the router.

### *5.4. ThingsBoard local server IoT platform*

ThingsBoard is an open-source IoT platform for data collection, processing, visualization, and device management [44]. It provides an out-of-the-box IoT cloud or on-premises solution to enable server-side infrastructure for various IoT applications [44]. Built on Java 8 platform, ThingsBoard provides 100 percent support for standard IoT protocols for device connectivity, including MQTT, CoAP, and HTTP(S), and it presently supports three different database options: SQL, NoSQL, and Hybrid databases. The ThingsBoard platform uses these databases to store *entities* (such as devices, assets, dashboards, users, alarms, customers, etc.), and *telemetry data* (attributes, time-series sensor readings, statistics, events, etc.). *Telemetries* are time-series of key-value pairs of data associated with a specific device, and ThingsBoard stores its received data as *telemetries*. *Assets* are containers for reorganizing the received data, and could be used to upload the results of the data processing. *Attributes*, on the other hand, usually represent the device features such as firmware version, hardware specifications, etc. which are assigned to registered devices and assets in the form of key-value pairs. The SQL database such as PostgreSQL stores all *entities* and *telemetry* in SQL database, while the

NoSQL option such as Cassandra stores all *entities* and *telemetry* in NoSQL database. In the Hybrid database option, all *entities* are stored in SQL database while all *telemetry* are stored in NoSQL database [11, 30, 38, 44]. In this project, the PostgreSQL database is installed on the ThingsBoard server for *entities* and *telemetry* storage.

ThingsBoard has two different editions, the Community Edition, which is free and wholly open source, and the Professional Edition, which has more advanced features. In this project, the Community Edition is used. This Community Edition is open source, and is available free-of-charge on both the ThingsBoard official website and on GitHub software development platform [44]. With the ThingsBoard back-end written in Java, and some of its micro-services based on Node.js, the ThingsBoard architecture is designed to be scalable, fault-tolerant, robust and efficient, customizable, and durable. The basic ThingsBoard architecture is shown in Figure 5 [30, 44].

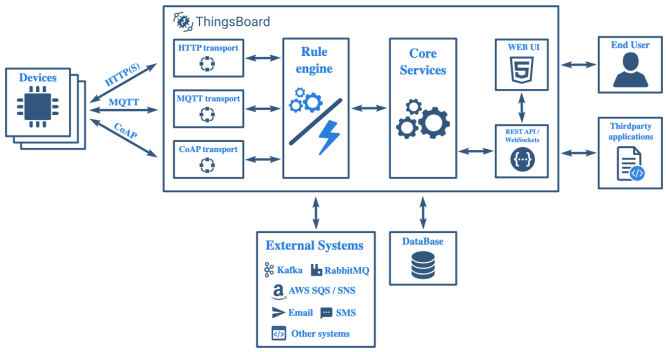


**Figure 5.** The basic ThingsBoard architecture.

The key features and functions of the major components of the ThingsBoard architecture are presented as follows;

- **Transport components:** These include MQTT, HTTP, and CoAP-based APIs for device applications and firmware. Being a part of the ThingsBoard "Transport Layer", each of the components here helps to push data to the *Rule Engine*, and could also use *Core Services* to issue requests to the database to validate device credentials [11, 30, 38, 44]. In this project, the MQTT-based device API supported by the MQTT communication protocol is implemented. MQTT is favoured ahead of the HTTP and CoAP because of its unique features like support for constrained resources such as low bandwidth, and it can be implemented over various TCP/IP connectivities. In addition, ThingsBoard server nodes act as an MQTT Broker that supports QoS levels 0 (at most once) and 1 (at least once), and a set of predefined topics which means that an external device can be configured as an MQTT Client to publish data to the server nodes [39]. Here, ESP32 is programmed and configured as that MQTT Client to publish the acquired PV System data to the ThingsBoard server nodes.
- **Rule Engine components:** The ThingsBoard *Rule Engine* helps in processing the incoming messages with user-defined logic and flow. This *Rule Engine*, which comprises of *Rule Node* and *Rule Chain*, is highly customizable and configurable, and can be used for processing complex events [30, 44]. For example, the system administrator is able to filter, enrich, and transform incoming messages sent by IoT devices and related applications, as well as implement and trigger various actions such as alarms, notifications on the state of the connected devices, email

alerts, or other communication with external devices using the *Rule Engine* [44]. In this work, the configured *Rule Engines* for the overall system implementation (e.g data transfer), and that for the alarm notifications on the state of the PV system being monitored (Storage Battery Voltage monitoring) are shown in Figures 6 and 7 respectively.

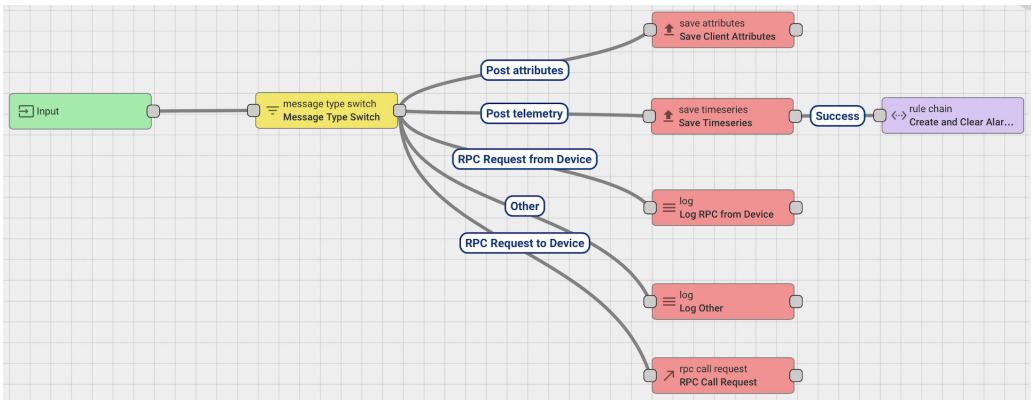


**Figure 6.** System Rule Engine.

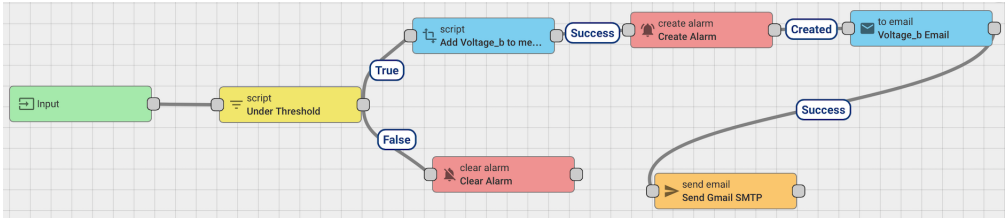


**Figure 7.** Alarm Rule Engine.

- **Core services:** These are responsible for handling REST API calls, monitoring device connectivity states, and WebSocket Subscriptions on entity, telemetry and attribute changes [44].
- **External systems:** Using the Rule Engine, communications can be established between ThingsBoard and external systems. This involves pushing data to external systems, processing the data, and reporting the results of the processed data back to ThingsBoard server for visualization [30,44]. Being able to integrate ThingsBoard data processing with external systems in this way ensures a great deal of flexibility in the system, unlike most IoT platforms [38].

ThingsBoard server can either be utilized directly on the *Live Demo* platform, installed on a private machine *On-Premise*, or hosted on a *Cloud Server* such as DigitalOcean, Amazon Web Services (AWS), Google Cloud platform, Microsoft Azure, IMB Cloud, etc [44]. The few literatures found on ThingsBoard have used the Live Demo platform to implement their proposed IoT-based monitoring solutions [11, 30]. This Live Demo platform requires the public internet for data access just like every other web application out there, which could leave the stored data vulnerable to internet attacks. On the other hand, hosting the ThingsBoard server on a Cloud platform such as AWS, requires not just the public internet for data access, it also requires subscriptions which might not be affordable depending on the stored data footprint, which means more cost and the possibility of internet attacks [44]. However, security in a SCADA system is a critical issue as attacks on the SCADA could

compromise the important company data stored in the cloud. Therefore, in our proposed IoT-based SCADA system design solution, the on-premise, self-hosted ThingsBoard server option is implemented. Although the ThingsBoard developers have provided installation options and guides for various machines and operating systems, including Windows, Linux (CentOS and Ubuntu), Raspberry Pi, Docker (Linux and MacOS), Docker (Windows), Maven, and Cluster Setup, the ThingsBoard server used in this work is installed on a Raspberry Pi 2 model B machine. This is because the Raspberry Pi is reliable, portable, and consumes relatively low power compared to the other options. This Raspberry Pi option is also completely free as it has been built from scratch on the Raspberry Pi using the installation guides on GitHub and ThingsBoard official website. Keeping the power consumption to the possible minimum is essential since the proposed SCADA system is meant to be in operation 24/7 for effective monitoring and supervisory control. By installing the ThingsBoard server on the Raspberry Pi machine connected to a private network (MUN Network, and local Wi-Fi Network), it can be operated with and without the public internet, depending on what configuration is chosen based on the desired security and flexibility of the operation. This represents a major contribution of this paper as no related works have been found where such measures were considered.

ThingsBoard currently supports various hardware platforms, including Arduino, ESP32, ESP8266, NodeMCU, Raspberry Pi, and LinkIt One, which makes it perfect for IoT applications [44]. Any of the chosen hardware can be connected to the ThingsBoard server, and two different authentication options supported by ThingsBoard, including Access Tokens and X.509 Certificates, can be implemented [44]. In this work, the TTGO ESP32 LoRa32 V1.0 hardware with OLED display screen is connected to the installed ThingsBoard server, and the Access Token device authentication option implemented. This means that by registering the ESP32 device on the ThingsBoard server platform, and assigning Access Token to the device, the connected sensor data acquired by the ESP32 device can be published to the server using both the server IP address (Raspberry Pi IP address), and the Access Token to identify the platform. The interface of the installed ThingsBoard server on the Raspberry Pi machine showing the IP address, and the numerous menus such as *Rule Chains, Customers, Assets, Devices*, and so on, for various actions is shown in Figure 8.

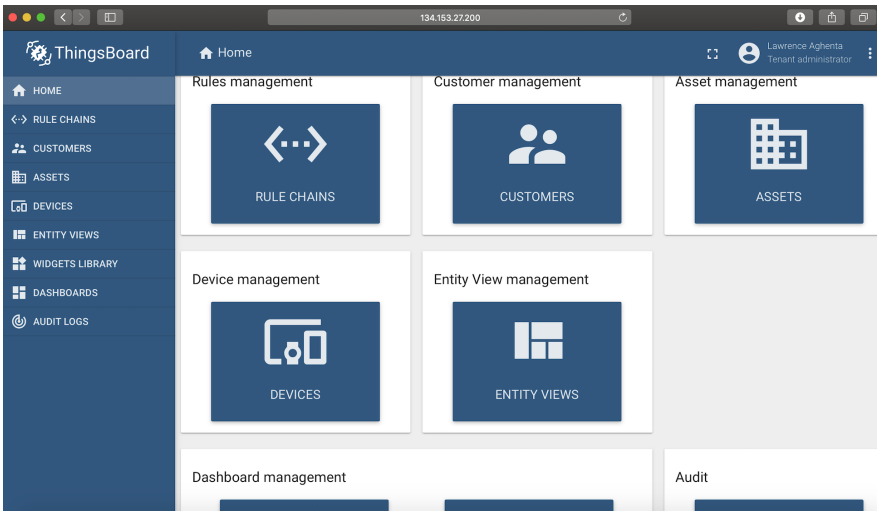


**Figure 8.** Raspberry Pi-installed ThingsBoard server interface.

This locally hosted ThingsBoard server on the Raspberry Pi single-board computer, which is the Master Terminal Unit (MTU) of the proposed SCADA system, serves as the MQTT Broker, and the ESP32 device, programmed and configured as an MQTT Client, publishes the acquired sensor data from the connected PV system to the MQTT Broker using MQTT protocol over the TCP/IP Wireless Network connectivity created with the Wi-Fi router. Since ThingsBoard allows for the creation of customizable real-time dashboards (HMIs) with more than 30 *Widgets* for data visualization, alarms, notifications, and device managements [30,44], beautiful real-time dashboards are created on the local ThingsBoard server for the PV system data visualization and management. These dashboards (HMIs) which show the states of the PV system being managed can be accessed either via the internet or offline, depending on the chosen configuration deployed. Figure 9 shows the connected ESP32 device, while Figure 10 shows the sensor data being posted to the ThingsBoard server nodes.

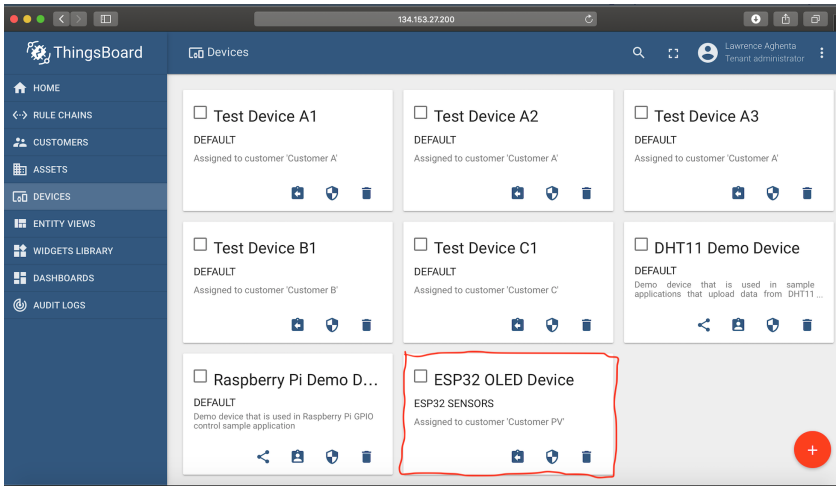


**Figure 9.** The connected ESP32 device.

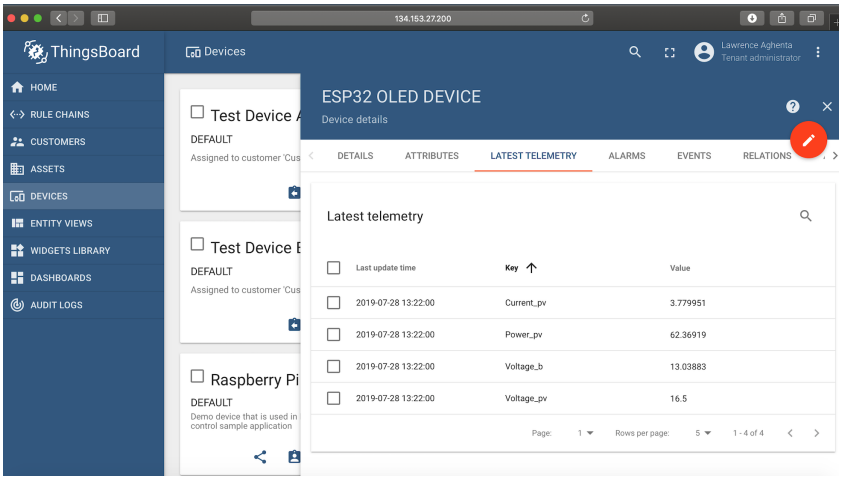


**Figure 10.** Sensor data posting.

### *5.5. MUN ECE laboratory PV system overview*

In order to test the functionalities of the designed open source SCADA system, it is setup to acquire the solar photovoltaic (PV) data (current, voltage and power) of the PV system at Memorial University (MUN) Electrical and Computer Engineering Department Laboratory. This PV system is made up of 12 Solar Panels covering a total area of 14 square meter and producing about 130 W and 7.6 A each, as well as Maximum Power Point Tracking (MPPT) controller and lead acid electrical battery system. Although the proposed SCADA system is connected to just one set of the modules (about 260 W, and 14 A output) for testing purposes, the entire system comprises of two modules connected in parallel such that it contains 6 sets of 260 W, and 14 A each.

## 6. Implementation methodology

In the proposed open source SCADA system design, the data and information flows between the solar PV system and the SCADA system are summarized in this section. First, the current, voltage, and power generated by the PV panels, and the storage battery voltage are measured and collected by the analog current and voltage sensors which are physically connected to the PV system using electrical wires. These data measurements and collection are made possible by using the developed Arduino IDE programs written and uploaded into the ESP32 micro-controller. Next, the ESP32 micro-controller, which is programmed and configured as an MQTT Client with the help of the PubSubClient MQTT Library, receives and processes these data from the sensors, and displays them on the Arduino IDE Serial Monitor and its OLED screen. Finally, the acquired PV system data are then published or transmitted via the MQTT Protocol over the locally created TCP/IP Wi-Fi connectivity to the self-hosted ThingsBoard IoT server platform, which is configured as an MQTT Broker. The published PV data received at the ThingsBoard server node are displayed as Telemetry messages. The server node is configured such that these data are automatically made available on the created dashboards and HMIs for remote monitoring and supervisory control actions. With respect to the QoS, data receipt acknowledgements are seen on the Arduino IDE Serial Monitor, and the published data in JSON format (Name: Value) are also displayed on the ESP32 OLED screen for local operator monitoring. The pseudocode describing this data (information) flow process is shown in Algorithm 1 below.

**Algorithm 1:** Data acquisition and logging algorithm:

Initialization;
1. Analog sensors measure and collect PV system data;
2. ESP32 reads sensor values on analog Pins 32, 34 and 35, and calculates values for Pins 32×34;
3. ESP32 displays the above values on Arduino IDE Serial Monitor and ESP32 OLED Screen;
4. ESP32 connects to local TCP/IP Wi-Fi Network with Wi-Fi Name and Password;
5. ESP32 MQTT Client identifies the local ThingsBoard IoT Server (MQTT Broker) via the Server IP Address;
6. ESP32 MQTT Client publishes sensor data to MQTT Broker over the TCP/IP Wi-Fi connectivity;
7. ThingsBoard Server displays data as Telemetry Messages on the specified Device using the Device Name and Access Token;
8. ThingsBoard Server Node logs the Telemetry Messages to Dashboards for data visualization;

**while** *ThingsBoard Server acknowledges data receipt* **do**
- 9. Display sensor data on ThingsBoard Server Node, Dashboards and ESP32 OLED Screen, and;
- 10. Display "DONE" on Arduino IDE Serial Monitor;
- **if** *No data receipt acknowledgement from ThingsBoard Server Node* **then**
  - 11. Display "FAILED.....retrying in 5 seconds" on Arduino IDE Serial Monitor;
- **else**
  - 12. Go to step 1;
- **end**

**end**

## 7. Prototype design

In this section, we describe the hardware implementation of the proposed IoT-based open source SCADA system solution. As shown in Figure 11, the Analog Current and Voltage Sensors are connected (via the pull-down resistors arrangement for the Current Sensor) to the TTGO ESP32 LoRa32 OLED device on a Breadboard using electrical wires. The inputs of the sensors are connected to the points of interest on the PV panel and storage battery system (PV System) using electrical wires such that the sensors measure and acquire the PV voltage, and current, and the storage battery voltage, as well as show the continuously calculated PV power from the PV voltage and current values. The power supplies for each of the components are provided as described in Section 5. The Wi-Fi Router used to setup the needed Wi-Fi connection, and the Raspberry Pi single-board computer hosting the ThingsBoard IoT server are both placed in the same building and integrated into the other components making up the SCADA system in the two different configurations described earlier.

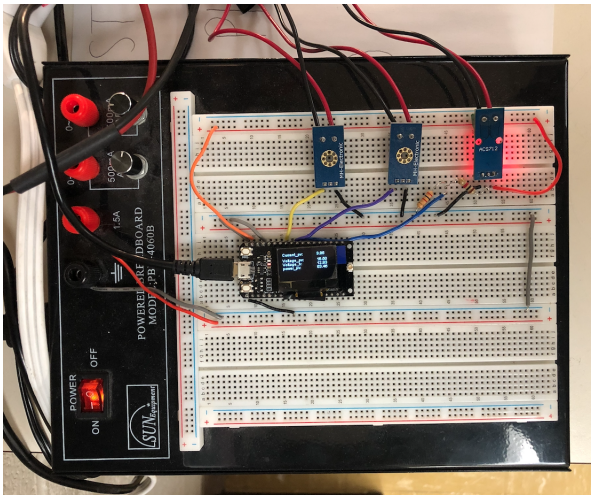


**Figure 11.** Hardware implementation of the proposed SCADA system.

## 8. Experimental setup of the proposed SCADA system

As described in Section 7 above, the hardware components were programmed, configured and setup for operation. The setup was then hooked up to the solar PV System in MUN ECE Laboratory. Figure 12 shows the analog sensors and ESP32 OLED device connected together and to the PV System, as well as some of the Dashboards created on the ThingsBoard IoT server platform (shown on the Laptop) for real-time data monitoring and supervisory control actions. As shown in the Figure, local operator monitoring of the acquired PV data is an additional feature in the proposed SCADA solution, and this is made possible using the ESP32 OLED Screen shown.

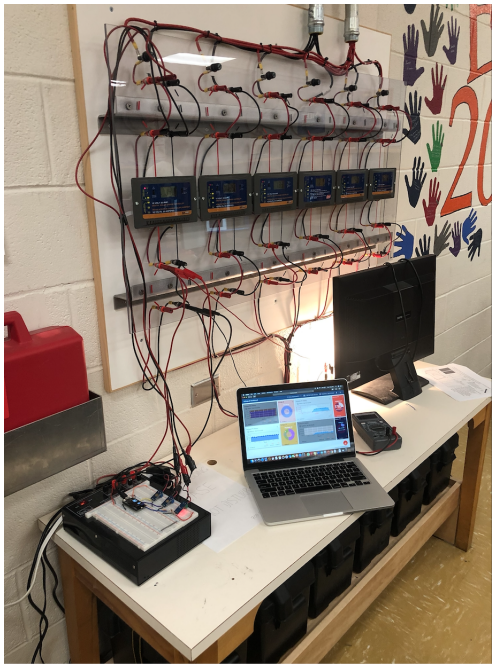

**Figure 12.** Experimental setup of the proposed SCADA system.

## 9. Testing and results

Having tested the proposed IoT-based open source SCADA system solution extensively by setting up the designed prototype to acquire, process, transmit, remotely monitor and initiate supervisory control actions of the MUN ECE Laboratory PV System data, we present the results and some of the created HMIs (Dashboards) in this section.

### *9.1. Results*

Each of the two hardware configurations, A and B (Figures 3 and 4 respectively) was set up and connected to the standalone solar PV system with the analog current and voltage sensors connected to the points of interest on the PV system to collect the desired data, the PV Current, the PV Voltage, the PV Power, as well as the storage battery Voltage after the MPPT system. The main data server locally installed on the Raspberry Pi machine and hosted on MUN Network, the ThingsBoard IoT server node (MQTT Broker), was configured to receive the sensor data being collected by the sensors, processed and published by the ESP32 micro-controller (MQTT Client). Data transfer was realized using the developed MQTT protocol over the local Wi-Fi network created. The real-time sensor data published as *key:value pairs*, and received as *latest telemetry data* on the *ESP32 OLED Device* are shown in Figure 10. The Figure also shows the time stamps of the *latest telemetry data* received.

At the ThingsBoard IoT server platform, dashboards were created for the remote monitoring of the solar PV data, and for easy data trends visualizations. Figure 13 shows multiple dashboards for the various PV system variables being acquired; the storage battery Voltage, the PV Current, Voltage and Power, such that the trends of each of the variables can be remotely monitored in one platform. As can be seen, the vibrations of the values of each of the variables were due to the weather conditions in St. John's at the various times of testing as expected since PV system outputs are affected by environmental conditions such as solar irradiance and temperature. At the time of logging these data, a digital multimeter was also used to locally measure each of the PV system variables so as to validate the accuracy of the acquired data seen on both the OLED display screen and at the ThingsBoard server platform. The acquired sensor values were found to be the same as those measured locally with the multimeter. Figure 14 shows a dashboard created to specifically test configuration A, while Figure 15 shows another dashboard specifically created to test configuration B. As seen in the figures, the sudden increase and decrease in the values (especially in Figure 14) happened at various times when the storage battery was being discharged with an electric load (a light bulb) connected across it. For example, discharging the storage battery made more current to flow from the PV panels to recharge the battery, thereby increasing the overall PV system outputs, including the current. The opposite effect also happened with the light bulb disconnected, until the PV system reached its stable point where the variables became mostly constant depending on the prevalent environmental conditions at that time. As expected, similar data values were recorded using both configurations A and B, with the values only affected by the prevalent environmental conditions at the time of testing. The major difference between the two configurations is the manner in which the recorded and stored data on the ThingsBoard server platform can be accessed as described earlier.

The proposed SCADA was tested extensively at various times of the day, and left connected to the PV system to continuously log the PV data for about a month so as to confirm the robustness of the designed system, and the results were found to be consistent with the locally measured values using a

digital multimeter, showing that the SCADA system performed optimally and accurately regardless of the environmental conditions and duration of testing. Also, as shown in Figures 11 and 12, the most recent data values were available for viewing on the ESP32 OLED display screen, thereby providing a local data monitoring interface whenever necessary.

Furthermore, in order to test the supervisory control capabilities of the proposed system, the *Rule Engine* tool of the ThingsBoard IoT server (Figures 6 and 7) was used to create a test alarm for the storage battery voltage values, and the alarm was configured such that for voltage values above 14 V, the alarm would get triggered to notify the system administrator of the current situation. For instance, the alarm was triggered automatically between 15:00 and 17:00 on the dashboard in Figure 16 when the voltage values rose to about 15 V. Also, although not presented here due to space constraint, a data table showing data history was also created at the ThingsBoard IoT server platform for easy data trends analysis.

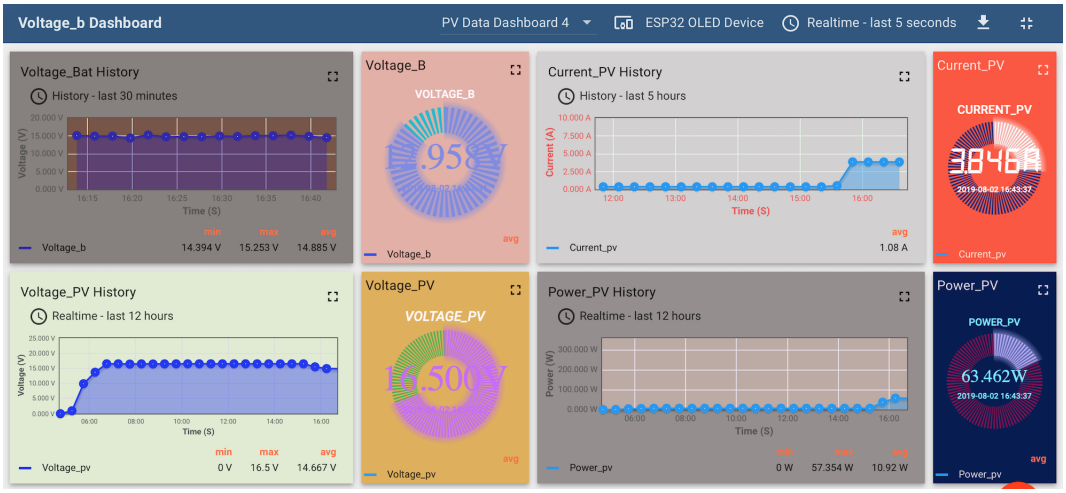


**Figure 13.** Created dashboards showing real-time data.

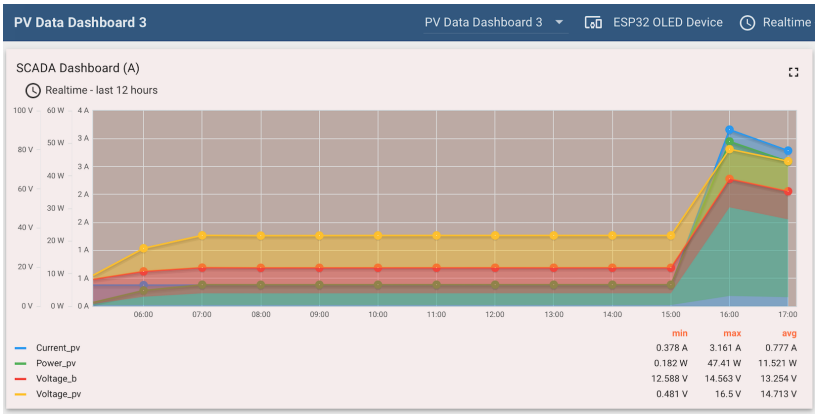


**Figure 14.** Created dashboard (A) showing real-time data.

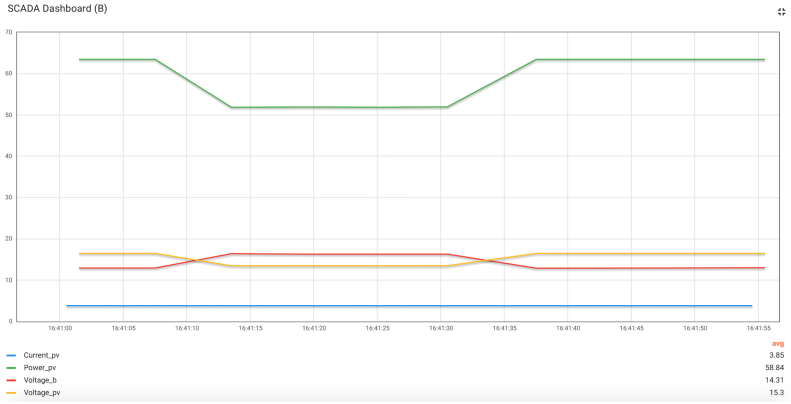


**Figure 15.** Created dashboard (B) showing real-time data.

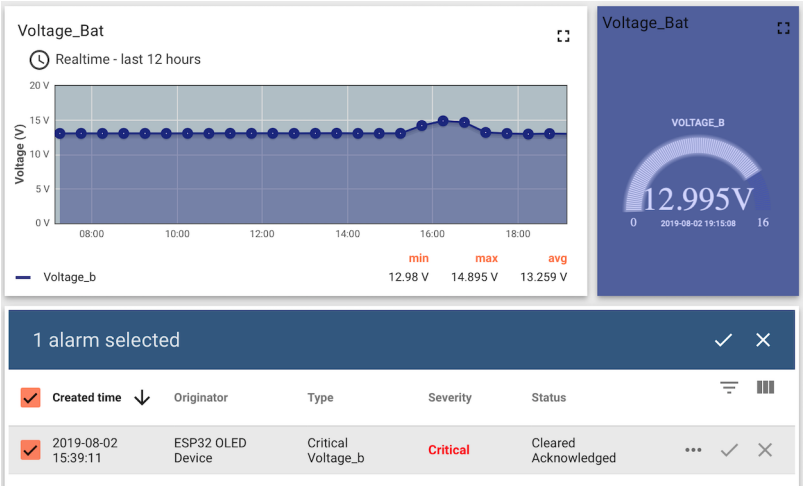


**Figure 16.** Test alarm.

## 10. Discussion

In this section, we briefly describe some of the key features of the designed IoT-based open source SCADA system solution based on the testing and the results realized from the testing:

- **Internet of Things based SCADA system:** The proposed open source SCADA system is based on the most recent SCADA architecture, the Internet of Things. The system features the four essential elements desirable in a SCADA system, including Field Instrumentation Devices (Current and Voltage Sensors), Remote Terminal Unit (TTGO LoRa32 ESP32 OLED Micro-controller), Master Terminal Unit (ThingsBoard IoT Server), and SCADA Communication Channel (MQTT data transfer protocol over Wi-Fi connectivity).
- **Data acquisition and historic storage:** With the SCADA system solution, data in any process plant of interest can be acquired for remote monitoring and storage. An SQL database, PostgreSQL, is installed with the ThingsBoard IoT server platform on the Raspberry Pi such that data can be stored for future use. As a result of this feature, historic data trends can be viewed on the server platform (e.g., Figure 13). Knowing the trends in any data sets could help decision

makers to make important business decisions.

- **Remote plant monitoring:** From the created dashboards for human machine interactions on the ThingsBoard IoT server platform, the current state of the process plant being managed can be monitored remotely in terms of data. Examples of such dashboards (HMIs) are presented above.
- **Local operator monitoring:** In the SCADA system solution proposed, local operator interface is considered. This is made possible using the OLED display screen on the ESP32 micro-controller where a local operator can view the most recent sensor data at the plant site simply by looking at the OLED screen. This could be particularly important in the events that the overall system administrator at the server end isn't available to provide system updates.
- **Reporting:** From the data logs, charts, alarms and data trends, reports on the state of the process plant can be generated for critical business decisions. In addition, reports on the states of the created alarms can be sent automatically to *Device Customers* and *System Administrator* either with instant messaging notifications or via emails.
- **Security:** The proposed SCADA system solution is such that the main data server, the ThingsBoard IoT server platform, is locally hosted on MUN network. As such, security measures can be taken by the system administrator to ensure data integrity. Such security measures include access control, firewalls, authorization, regular risk assessment, whitelists, continuous monitoring and log analysis, updating and patching regularly, authentication, and so on.
- **Reliability and availability:** Even though system reliability calculation is outside the scope of this paper, studies have shown that SCADA system reliability and availability are affected by delayed or wrong operator decisions [45]. In the proposed open source SCADA system solution, all the components are open source and readily available. Also, the main cloud server for data processing is locally installed on MUN Network (hosted), and self-managed. Therefore, the system administrator is always available to continuously manage and maintain the system in order to ensure its continuous reliability and availability. However, this is almost impossible in a commercial (proprietary) SCADA system where the customer is beholden to a single vendor, and as such has to contact the vendor in the events of the SCADA system failures which could lead to downtime.
- **Supervisory control:** In the proposed SCADA system solution, by remotely monitoring the data sets of the desired variables in the process plant, the system administrator can send supervisory control commands via emails, alarms or instant messaging notifications to the local operator informing the operator of the problems and requesting immediate actions to correct the problems.
- **Ease of use:** The main data server, ThingsBoard IoT server platform, where data processing and human machine interactions are carried out by the system owner is simple and user-friendly, meaning that it requires less customer training for continuous use. This is usually not the case in most commercial SCADA system solutions as their MTU platforms are complicated, requiring a great deal of training and experience for operation.
- **Open source, and low cost:** In the proposed SCADA system solution, all the components are manufactured and supplied by multiple vendors (mix and match), and are readily available under open source license. The components can easily be interconnected with related components from several vendors. As such, the consumer is not beholden to a single vendor. This is one of the key features of an open source system. In Table 1, it can be seen that despite the fact that the

proposed SCADA system performs all the basic functions desired in a SCADA system, it is a low-cost solution compared to the available commercial SCADA systems which are in thousands of dollars. The overall system cost is just about $280 CAD. From the Table, it can be seen that the main data server, the ThingsBoard IoT server installed on the Raspberry Pi, and the database software, PostgreSQL, cost nothing. This is because the server was built from the source code alongside the database using the developers' guides. The MQTT Client Library (PubSubClient) on Arduino IDE is also free.

**Table 1.** Bill of materials.

| S/N | COMPONENT | QTY | PRICE (CAD) |
|---|---|---|---|
| 1 | ThingsBoard IoT Server | 1 | 00.00 |
| 2 | PostgreSQL. | 1 | 00.00 |
| 3 | Raspberry Pi 2 B | 1 | 45.95 |
| 4 | TTGO LoRa32 ESP32 OLED | 1 | 17.49 |
| 5 | 16GB Memory Card | 1 | 19.99 |
| 6 | Voltage Sensor | 2 | 11.98 |
| 7 | Current Sensor | 1 | 5.25 |
| 8 | D-Link D1-524 Wireless Router | 1 | 98.51 |
| 9 | Miscellaneous (Boxes, Breadboard, Resistors, Wires, etc.) | 1 | 80.00 |
| | **Grand Total:** | | **$279.17 CAD** |

- **Low power:** For an IoT-based system designed to operate 24/7, power consumption is a major factor in selecting the individual system components. Hence, power consumption was a major factor considered in choosing the components used in the proposed SCADA system design. The power consumption of the individual components were measured simultaneously using Kilowatt meters while the system was in operation to ascertain the power consumption of the overall SCADA system solution. As can be seen in Table 2, the overall power consumption of the system is about 9.3 W. Although this overall power consumption is relatively low, it could still be reduced since the major components, the Raspberry Pi (ThingsBoard IoT server) and ESP32 OLED micro-controller, only consume a combined total of 2.7 W while in operation, which is relatively low. The buck of the overall power consumption value is mostly due to the high power demands of both the D-Link Wi-Fi Router and the Breadboard, which could easily be eliminated or replaced with similar components requiring less power. For instance, the D-Link Wi-Fi Router can be eliminated by configuring the Raspberry Pi as a Wireless Access Point to provide the needed Wi-Fi connectivity, while the breadboard can be replaced with a smaller breadboard requiring less power.

**Table 2.** Power consumption of hardware components.

| S/N | HARDWARE | POWER(W) |
|---|---|---|
| 1 | Raspberry Pi 2 B | 1.8 |
| 2 | ESP32 OLED (alone) | 0.9 |
| 3 | D-Link D1-524 Wireless Router | 4.2 |
| 4 | Breadboard (with ESP32 OLED, Sensors, Resistors, etc. connected) | 3.3 |
| | **Overall System Power Consumption (less ESP32 OLED alone):** | **9.3 W** |

## 11. Conclusions

With most companies' critical assets spread over large geographical areas, sometimes in harsh environments, especially hybrid power system components comprising of both the conventional generation sources such as fossil fuels, and sustainable (renewable) generation sources such as solar photovoltaic (PV) systems and wind turbines, it is imperative to have a flexible, secure, cost-effective and reliable coordinated means of overseeing the operations of the various generation sources, in addition to a local monitoring interface. Despite the fact that SCADA systems have made such coordinated monitoring and control of these distributed assets possible, SCADA system design solutions, implementations and deployments have largely remained proprietary as these solutions are mostly developed by automation companies across the globe. However, the high costs of these proprietary SCADA systems are hugely unjustifiable for smaller applications. In addition to these high costs, there is also the issue of the SCADA system interoperability with the existing hybrid power system infrastructures which are usually from multiple manufacturers and suppliers. Such infrastructures include energy storage systems, communication systems, power electronic converters, and so on. Therefore, an open source SCADA system represents the most flexible and most cost-effective SCADA system solution, especially for smaller applications. With an open source SCADA solution, the user is not beholden to a single vendor, and is able to combine components from multiple vendors under open source licenses to ensure the system interoperability with existing infrastructures and reduction in the overall system cost.

In this paper, we proposed a low-cost open source SCADA system based on the most recent SCADA architecture, the Internet of Things (IoT). We also demonstrated the hardware implementation of our proposed SCADA system solution using very few low-cost, low-power, open source and readily available components as the essential elements of the SCADA system. In designing our proposed open source SCADA system solution, data security, data integrity, and system reliability were taken into consideration since security in a SCADA system is a critical issue. These considerations were implemented by locally installing the main data server, the ThingsBoard IoT server, on a Raspberry Pi single-board machine. Thus, the data server was locally hosted and self-managed on MUN Network such that data security and data integrity measures like authentication, authorization, access control, whitelisting, log analysis, and firewalls are self-managed by the system administrator to ensure data security, data integrity, and system availability, and thus making sure that the system is reliable. Also, we showed the use of the lightweight IoT application protocol, MQTT protocol, for data transmission in such applications. The overall SCADA system cost was found to be extremely low, about $280 CAD, and the overall power consumption while in

operation was found to be minimal, about 9.3 W. We also demonstrated the performance of our proposed open source SCADA solution by setting it up to collect, log, process, and remotely monitor the current, voltage and power of a standalone 260 W, 12 V solar photovoltaic (PV) system. Supervisory control actions were also considered with alarms created to trigger notifications in the events that the storage battery voltage was above a certain set point. From our testings and results, we showed that the proposed open source SCADA system operates properly and accurately. With the OLED display screen of the ESP32 micro-controller board used, a local real-time data monitoring interface was also incorporated into the proposed SCADA system solution.

Even though the proposed open source SCADA system has only been tested with a standalone solar PV system in this work, the system can also be customized for use in other applications requiring real-time data acquisition, remote monitoring and supervisory control such as traffic signal systems, power transmission and distribution systems, mass transit systems, home energy management systems, and so on.

## 12. Future work

ThingsBoard IoT server supports other important alarm types. For example, alarms can be configured to notify the system administrator when the process plant being monitored is offline, and alarm notifications can be sent via emails directly to the customers assigned to that particular device and the system administrator. As a future work, we will look at incorporating these alarm types into the system to increase the functionalities of the system. Furthermore, data encryption can be implemented on the communication channel and data transfer protocol for better security.

## Acknowledgments

The authors would like to thank the School of Graduate Studies, Faculty of Engineering and Applied Science, Memorial University and the Natural Sciences and Engineering Research Council of Canada (NSERC) Energy Storage Technology Network (NESTNet) for providing the necessary funds and the conducive environment to carry out this research work. The authors would also like to acknowledge the technical and emotional supports of friends and families throughout the difficult period of carrying out this research.

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Energy Storage Technology Network (NESTNet).

## Conflict of interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

## References

1. Aghenta LO and Iqbal MT (2019) Design and Dynamic Modelling of a Hybrid Power System for a House in Nigeria. *Int J Photoenergy* 2019: 1–13.
2. IEC White Paper (2019) Electrical Energy Storage. Available from: `https://www.iec.ch/whitepaper/pdf/iecWP-energystorage-LR-en.pdf`.

3. Lee J, Lee S, Cho H, et al. (2018) Supervisory Control and Data Acquisition for Standalone Hybrid Power Generation Systems. *Sustainable Computing: Informatics and Systems* 20: 141–154.
4. Stouffer K, Falco J and Kent K (2011) Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security—Recommendations of the National Institute of Standards and Technology. Special Publication 800-82.
5. Jiao D and Sun J (2018) Real-Time Visualization of Geo-Sensor Data Based on the Protocol-Coupling Symbol Construction Method. *ISPRS Int J Geo-Inf* 7: 460.
6. Lu X (2014) Supervisory Control and Data Acquisition System Design for CO2 Enhanced Oil Recovery. Master of Engineering Thesis, Technical Report No. UCB/EECS-2014-123. EECS Department, University of California at Berkeley.
7. Sajid A, Abbas H and Saleem K (2016) Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges. *IEEE Access* 4: 1375–1384.
8. Al-Fuqaha A, Guizani M, Mohammadi M, et al. (2015) Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun Surv Tut* 17: 2347–2376.
9. Sethi P and Sarangi SR (2017) Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering* 2017: 1–25.
10. Nicola M, Nicola C, Duta M, et al. (2018) SCADA Systems Architecture Based on OPC and Web Servers and Integration of Applications for Industrial Process Control. *International Journal of Control Science and Engineering* 8: 13–21.
11. Alavi SA, Rahimian A, Mehran K, et al. (2018) An IoT-Based Data Collection Platform for Situational Awareness-Centric Microgrids. *2018 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)* 1–4.
12. Kao K, Chieng W and Jeng S (2018) Design and development of an IoT-based web application for an intelligent remote SCADA system. *2018 IOP Conference Series: Materials Science and Engineering* 323.
13. Li W, Wang J, Yen C, et al. (2018) Cloud supervisory control system based on JustIoT. *2018 IEEE International Conference on Smart Manufacturing, Industrial and Logistics Engineering (SMILE)* 17–20.
14. Sarierao BS and Prakasarao A (2018) Smart Healthcare Monitoring System Using MQTT Protocol. *2018 3rd International Conference for Convergence in Technology (I2CT)* 1–5.
15. Wu F, Wu T and Yuce M (2018) An Internet-of-Things (IoT) Network System for Connected Safety and Health Monitoring Applications. *Sensors* 19: 21.
16. Yi D, Binwen F, Xiaoming K, et al. (2016) Design and implementation of mobile health monitoring system based on MQTT protocol. *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* 1679–1682.
17. Kodali RK and Soratkal S (2016) MQTT based home automation system using ESP8266. *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)* 1–5.
18. Bassoli M, Bianchi V and Munari I (2018) A Plug and Play IoT Wi-Fi Smart Home System for Human Monitoring. *Electronics* 7: 200.

19. Chang CY, Kuo CH, Chen JC, et al. (2015) Design and Implementation of an IoT Access Point for Smart Home. *Applied Sciences* 5: 1882–1903.

20. Lee Y, Hsiao W, Huang C, et al. (2016) An integrated cloud-based smart home management system with community hierarchy. *IEEE T Consum Electr* 62: 1–9.

21. Pirbhulal S, Zhang H, Alahi ME, et al. (2017) Erratum: Sandeep P., et al. A Novel Secure IoT-Based Smart Home Automation System Using a Wireless Sensor Network. *Sensors* 17: 69.

22. Sahadevan A, Mathew D, Mookathana J, et al. (2017) An Offline Online Strategy for IoT Using MQTT. *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)* 369–373.

23. Mishra B (2018) TMCAS: An MQTT based Collision Avoidance System for Railway networks. *2018 18th International Conference on Computational Science and Applications (ICCSA)* 1–6.

24. Kodali RK (2016) An implementation of MQTT using CC3200. *2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)* 582–587.

25. Dow C, Cheng S and Hwang S (2016) A MQTT-based guide and notification service system. *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)* 1–4.

26. Bryce R, Shaw T and Srivastava G (2018) MQTT-G: A Publish/Subscribe Protocol with Geolocation. *2018 41st International Conference on Telecommunications and Signal Processing (TSP)* 1–4.

27. Dhar P and Gupta P (2016) Intelligent parking Cloud services based on IoT using MQTT protocol. *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)* 30–34.

28. Muladi M, Sendari S and Widiyaningtyas T (2018) Outdoor Air Quality Monitor Using MQTT Protocol on Smart Campus Network. *2018 International Conference on Sustainable Information Engineering and Technology (SIET)* 216–219.

29. Atmoko RA and Yang D (2018) Online Monitoring and Controlling Industrial Arm Robot Using MQTT Protocol. *2018 IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics)* 12–16.

30. De Paolis LT, De Luca V and Paiano R (2018) Sensor data collection and analytics with thingsboard and spark streaming. *2018 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)* 1–6.

31. Pesch A and Scavelli P (2019) Condition Monitoring of Active Magnetic Bearings on the Internet of Things. *Actuators* 8: 17.

32. Reaves B and Morris T (2012) An open virtual testbed for industrial control system security research. *Int J Inf Secur* 11: 215–229.

33. Hadžiosmanović D, Bolzoni D and Hartel PH (2012) A log mining approach for process monitoring in SCADA. *Int J Inf Secur* 11: 231–251.

34. Unique Automation Portfolio. Available from: `https://new.siemens.com/ca/en/products/automation`.

35. Sultana T and Wahid KA (2019) Choice of Application Layer Protocols for Next Generation Video Surveillance Using Internet of Video Things. *IEEE Access* 7: 41607–41624.

36. Moustafa N, Turnbull B and Choo KR (2019) An Ensemble Intrusion Detection Technique Based on Proposed Statistical Flow Features for Protecting Network Traffic of Internet of Things. *IEEE Internet of Things Journal* 6: 4815–4830.

37. Babovic ZB, Protic J and Milutinovic V (2016) Web Performance Evaluation for Internet of Things Applications. *IEEE Access* 4: 6974–6992.

38. Ismail AA, Hamza HS and Kotb AM (2018) Performance Evaluation of Open Source IoT Platforms. *2018 IEEE Global Conference on Internet of Things (GCIoT)* 1–5.

39. ThingsBoard API Reference. Available from: `https://thingsboard.io/docs/reference/mqtt-api/`.

40. Nuratch S (2018) Applying the MQTT Protocol on Embedded System for Smart Sensors/Actuators and IoT Applications. *2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)* 628–631.

41. Aghenta LO and Iqbal MT (2019) Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing. *Electronics* 8: 822.

42. ESP32 TTGO. Available from: `http://esp32-ttgo.blogspot.com`.

43. Zhong X and Liang Y (2016) Raspberry Pi: An Effective Vehicle in Teaching the Internet of Things in Computer Science and Engineering. *Electronics* 5: 56.

44. ThingsBoard Documentation. Available from: `https://thingsboard.io/docs/`.

45. Nasr PM and Yazdian-Varjani A (2018) Toward Operator Access Management in SCADA System: Deontological Threat Mitigation. *IEEE T Ind Inform* 14: 3314–3324.


AIMS Press