Implementation of a Fully-automated Optimized Fog-computing-based IoT-controlled PV Network

Adel M. AlWaqfi
Electr. Eng. Dept.
Al Hussein Technical University
Amman, Jordan
email: adelalwaqfy@gmail.com

Mohammad J. Abdel-Rahman
Data Sci. Dept., Princess Sumaya University for Technology
Electr. Eng. Dept., Al Hussein Technical University
Amman, Jordan
0000-0001-5788-6656
email: m.AbdelRahman@psut.edu.jo

Yunis A. Al-Qreenawi
Electr. Eng. Dept.
Al Hussein Technical University
Amman, Jordan
email: yalqreenawi@gmail.com

Zain AlHwaidy
Electr. Eng. Dept.
Al Hussein Technical University
Amman, Jordan
email: zainalhwaidy@gmail.com

Ahmad I. Abu-El-Haija
Electr. Eng. Dept.
Jordan University of Science and Technology
Ar Ramtha, Jordan
email: haija@just.edu.jo

***Abstract*—The Internet of Things (IoT) is revolutionizing almost every aspect of our lives. Smart grids are one example of this. Integrating IoT into various fields is a challenging task. In this paper, we present a detailed implementation of a fog-computing-based IoT system for monitoring and controlling a photovoltaic (PV) power network. As a case study, the implemented system is used to facilitate automatic energy routing within two PV systems feeding different loads. Our results demonstrate the ability of our IoT system to efficiently and automatically monitor and control a network of PV systems.**

Keywords- Photovoltaic (PV) systems; IoT; fog computing; automation.

I. INTRODUCTION

The Internet of Things (IoT) is a collection of technologies that grant us the ability of interfacing almost any physical object in the world, that is of interest in terms of its data and actions, to the digital world of computing, making it possible to monitor, analyze, and control mechanical and electrical systems everywhere, as well as driving insights about those connected things. The research and implementations in fields such as energy, healthcare, retail, transportation, and agriculture are continuously emerging and coming into reality in the form of smart gadgets, smart homes, smart factories, smart grids, self-driven cars, and smart cities [1][2].

In essence, the availability of real-time sensory data and sufficient computing resources constitutes the real power of IoT. However, the huge amount of data that will be generated in the process of digitizing the world requires deliberate design of the underlying infrastructures and technologies that are responsible for data delivery, processing, and storage. Here comes the concepts of cloud, fog, and edge computing into play [3]. Cloud computing can provide the necessary hardware and software for hosting, managing, and operating IoT solutions in the form of Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Nevertheless, edge and fog computing concepts are believed to be supplemental technologies that become necessary when the latency of delivering the solution is crucial or local (i.e., near the source of data) awareness of the solution is required.

Wireless Sensor Networks (WSNs) can be integrated into the Internet to move the currently connected networks from the local to the global domain [4][5], which essentially extends the landscape of IoT. Also, due to the fact that IoT applications are meant to be data-centric, there is a need for IoT and WSN communication protocols that are well-designed for this purpose. Existing application-layer IoT protocols in the literature are AMQP, CoAP, XMPP, MQTT, and HTTP. Other protocols that cover different layers of the OSI model are BLE, WiFi, Sigfox, NB-IoT, Zigbee, Zwave, and 6LoPWAN [6].

One of the most important applications of IoT is the Smart Grid (SG). SG is an electrical power network characterized by an infrastructure that makes it capable of sensing, communicating, monitoring, and controlling all parts of the network and addressing problems when occurring, unlike traditional networks [7]. In other words, a conventional grid can be transformed into a smart one with the help of IoT [8]. Renewable Energy (REs) sources such as wind and photovoltaic are considered the main and the most important generation sources relied upon in the smart microgrid. The stochasticity and unpredictability of the REs power generation make the smart grid nature potentially unstable. This brings the need for the IoT network to monitor the time-varying behavior of the generation and the load demand. All those challenges in the power system can be solved by integrating IoT. On top of that, comes the optimization layer that aims at minimizing the total cost of the microgrids by finding the optimal configuration between sources and loads based on real-time data provided by the smart sensors and reported by the IoT network.

***Our Contributions*—**Our main contributions are:

 ISBN: 978-1-68558-054-4

- We present a detailed implementation of an IoT solution to monitor and control a PV system starting from the lowest level of physical components (i.e., the sensor node), going through the communication, all the way to the fog node setup.
- We evaluate the setup by running a simple optimization algorithm on top of the collected data.

Paper Organization—The rest of the paper is organized as follows. In Section II, we review the previous works in the literature on IoT implementations for smart grids. In Section III, we explain in detail the steps involved in implementing each component of our setup. Preliminary results are presented in Section IV. Finally, our conclusions are summarized in Section V.

II. LITERATURE REVIEW

The potential gains and challenges of leveraging technologies like AI, IoT, and 5G in smart grids were discussed in [9]. The potential improvements in the system's robustness, reliability, resiliency, and security are faced with challenges such as the unreliable wireless channels used in IoT. In this section, we highlight a few papers that considered implementing IoT networks for smart grids.

In [10], the authors focused on the WSN part of the IoT stack, designed and implemented a new WSN platform that consists of power-independent nodes. The nodes employ rechargeable batteries using embedded photovoltaic batteries harvesting the ambient light starting from 100 lux. The authors in [11] presented an implementation for monitoring regular domestic conditions, such as temperature and light intensity. They used a ZigBee-based WSN, where the gateway of the network is responsible for translating the ZigBee format of data into an Internet IPV6 format before reporting it to a MySQL database on a Windows-based server and displaying the results using PHP and JavaScript on a browser. A WiFi-based WSN is implemented in [12] for monitoring and controlling environmental, safety, and electrical parameters of a home. The authors supplemented their stack with an Android application instead of a web-based approach. An end-to-end, dynamic, and scalable solution was implemented in [13]. The authors in [13] assume that all IoT devices are WiFi-capable and can communicate using MQTT. They used an ESP8266 microcontroller and a cloud-hosted server with a MongoDB database to store the data.

Optimizing the energy production, consumption, and storage in smart grids over time helps save costs and reduce energy wastage. IoT systems play a key role in automating and interconnecting equipment. As discussed in [14], the authors used an energy management algorithm to match the required load with the generated energy from PV and wind energy systems.

To the best of our knowledge, this paper presents the first solution that (i) combines the locality and reliability of RF-based WSNs with the powerful edge of a gateway and MQTT communications and (ii) leverages the structured nature of SQL database to store non-sensory data along with the dynamic properties of a time-series database (that are suitable for IoT data) and harness this to create a base for developing smart grid solutions. Our paper provides an end-to-end solution that illustrates the whole cycle of IoT, from data generation to response and actuation, without missing the middle parts of provisioning WSN, channeling and storing data, and algorithm deployment.

III. IMPLEMENTATION SETUP

We consider a multi-microgrid system that is managed locally (adopting a fog computing paradigm) by an IoT-based control system. Our system consists of three main components: A power system, a control system, and a communication system.

The implemented setup is designed to represent the considered IoT-controlled smart grid on a small scale, to test the feasibility of applying control algorithms to the power system. A stand-alone multi-microgrid system is built, consisting of two PV arrays with their inverters and batteries. We integrated IoT capabilities into the system by using sensor nodes, an edge node, and a fog node. The complete system with its three main components, power, control, and communication, is illustrated in Figure 1. Our implementation aims to have the best assignment of connections between microgrids to achieve reliable and available systems with the minimum cost. The components of the setup are as follows:

A. Power System

Two stand-alone PV systems, each with an AC capacity of 5.5 kWp were constructed at Al Hussein Technical University (HTU), Amman, Jordan. Each system consists of:

- Three stings with two polycrystalline Jinko panels, with a rated power 315 W (model type JKM315P) connected in series, south oriented with a slope of 24 degrees.
- Four 12 V-DC NPP batteries connected in series.
- A 5.5 kWp rated power MUST stand-alone inverter with a charge controller (model type PV18-5548VHM).
- AC cables 2C $\times 2.5$ mm^2 (CU/PVC/PVC) connected from the inverters to the loads.
- As a control device, a solid-state relay (model SSR-40DA).

B. Sensor Nodes (IoT Devices)

A sensor node is a combination of a Microcontroller Unit (MCU), a group of sensors, and a means to communicate its data with other nodes in the network or with a gateway. It can be also called an IoT device or a thing in the scope of IoT. IoT devices are generally designed to be resource-constrained in terms of energy and computational power. Thus, the integration between the sensor nodes and the fog layer is preferred to be through an edge node, which is not resource-constrained, can perform complicated tasks, and is equipped with advanced interfaces. In this case, the complexity needed for the fog layer communication is moved from each sensor node to the edge node. In our setup, multiple Arduino UNO microcontrollers serve as the mind of the IoT devices used to interface various deployed sensors. In addition, the UNO is

 ISBN: 978-1-68558-054-4

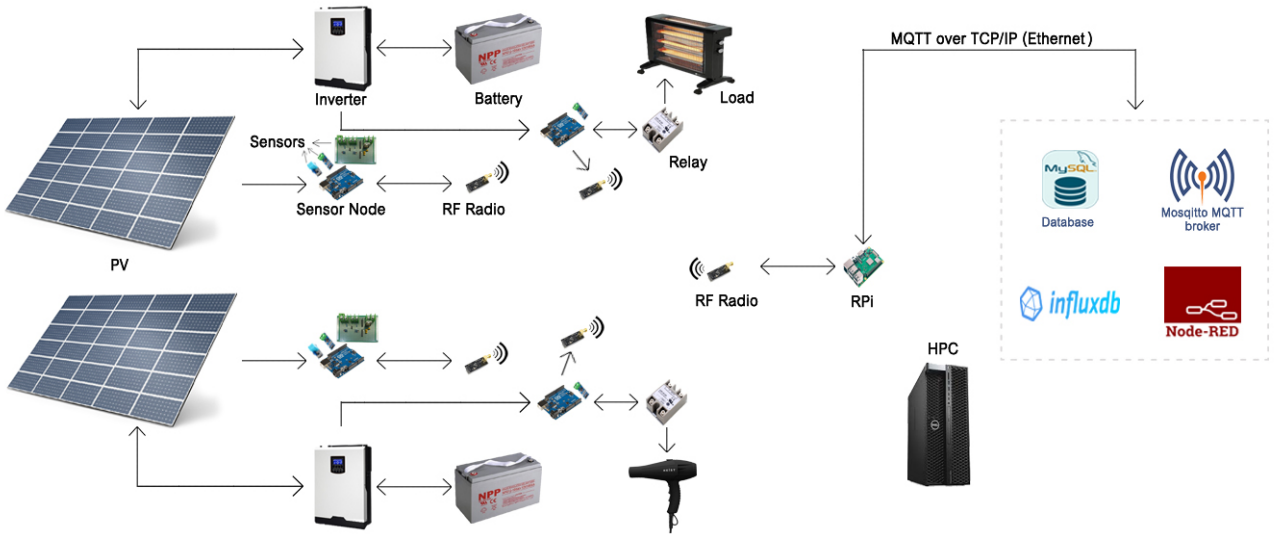


Figure 1. Implementation setup.

responsible for triggering the actuation process as a result of receiving commands in the IoT network. The microcontroller is based on an ATmega328 processor. It can support analog and digital sensors, as well as I2C and SPI sensor communication protocols. Usually, a node that performs only sensing tasks is called a sensor node or S node, whereas a node that performs both sensing and actuating tasks is called an SA node. We deploy a total of four nodes: three S nodes and one SA node. Two of the S nodes are used to read data from each inverter and one is connected to the load lines to measure their drawn currents. The SA node is used to control the assignment of sources to loads when needed.

C. Sensors and Actuators

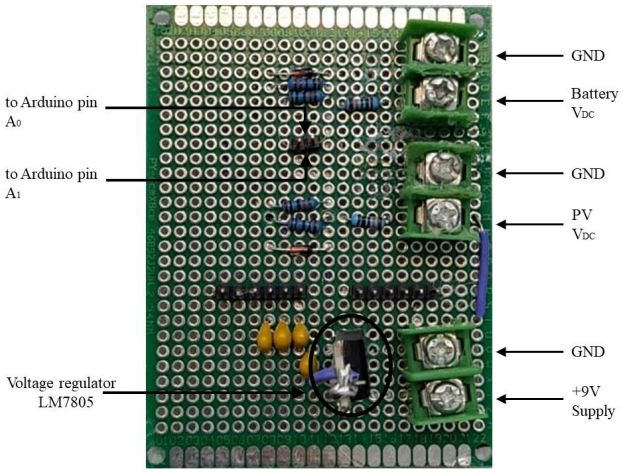


Figure 2. Voltage divider-based DC voltage sensor.

Multiple off-the-shelf sensors are used to comprise different sensor nodes in our system. They are used to measure quantities of interest for our use case. Those sensors are for measuring the electrical current and voltage of the DC output of the PV cells, the DC voltage of the batteries, and the AC output of the inverters.

For the current, we used ACS712 to measure the DC current output of the PV cells, the DC charging and discharging currents of the batteries, and the AC current output of the inverters. To measure the DC voltages of the PV output and the batteries, we designed a simple voltage divider-based sensor, shown in Figure 2. The sensor takes a DC input up to 50 V, converts it linearly into the 5 V range that the microcontroller can understand, and computes the voltage back. It uses the LM7805 linear voltage regulator to generate the needed reference voltage. For measuring the AC voltage, we used a transformer-based sensor, ZMPT101B. All the sensors have an analog output with a certain linear transfer function for a given range, which should be matched with the expected values to be measured. For actuating, we used solid-state relays (SSR-40DA) to control the contribution of each source (inverter) to each load.

D. Wireless Sensor Network

The communication inside the WSN is established using a wireless communication transceiver module nRF24L01+. The transceiver module is designed to operate in the worldwide ISM frequency band, and it uses GFSK modulation for data transmission. The center operating frequency is 2.4 GHz with 1 MHz channel bandwidth. With its reconfigurable transfer data rates (i.e., 250 Kbps, 1 Mbps, and 2 Mbps), it can serve the purpose of local wireless communication in the WSN. For an IoT application, 250 Kbps is sufficient to achieve the required performance. Moreover, the module comes with an implemented tree-topology network, in which using a single frequency channel out of the 125 available, can theoretically produce a network of up to 3,125 nodes. In Figure 3, we show the organization of the sensor nodes utilizing the tree topology, where each node is associated with a hexadecimal address to define its existence in the WSN. At the top of the network resides the gateway with an address of 00. Other sensor nodes are then assigned different addresses according

 ISBN: 978-1-68558-054-4

to the tree topology. There are five depth levels, each can support up to five nodes. For example, the first level contains the nodes $01, 02, 03, 04$, and 05. Under node 01, other five nodes can exist with the addresses $011, 021, 031, 041$, and 051. Message Queuing Telemetry Transfer (MQTT) is an IoT protocol for transferring data in the application layer and was originally developed by IBM. MQTT is a lightweight and flexible asynchronous protocol, which means that it can be implemented on heavily-constrained hardware and limited-bandwidth networks. In MQTT, a subscriber is a client that needs to communicate with other clients. The broker in the protocol serves as the central entity that manages all connections and pipelines between clients based on the publish/subscribe model. Compared to other IoT protocols, such as CoAP, XMPP, and AMQP, MQTT is more suitable for resource-constrained environments. Also, in MQTT, the broker hides the complexity of communication rather than being on the side of the clients [15][16]. We use MQTT for communication between the edge node and the fog node.

The deployed nodes, shown in Figure 3, can be explained in details as follows:

- **Node 01 (PV1) and Node 02 (PV2)**: Each contains a UNO MCU, an LM7805, three ACS712, a DC voltage-divider sensor, a ZMPT101, and an nRF24L01+.
- **Node 03 (Loads)**: Contains a UNO MCU, four ACS712, and an nRF24L01+.
- **Node 04 (Actuator)**: Contains a UNO MCU, eight SSR-40DA, and an nRF24L01+.

Note that we have four electrical paths, one from each source to each load. Two relays need to be deployed for each path, one for the line and the other for the neutral. Hence, we have a total of $4 \times 2 = 8$ relays in Node 04. When a path is opened or closed, the two relays should be actuated at the same time to ensure proper flow of power and avoid problems.

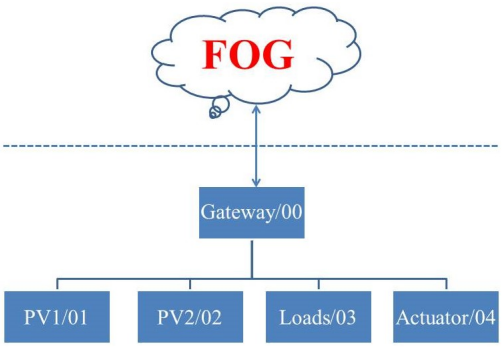


Figure 3. WSN tree topology.

E. Edge Node (Gateway)

Raspberry Pi (RPi) 4 is used to implement an MQTT client that bridges the local network of IoT devices with the MQTT broker running on the fog node. RPi 4 comes with a Quad-core Cortex-A72 (ARM v8) 64-bit CPU, 2GB of SDRAM, support of IEEE 802.11ac WiFi, Bluetooth 5.0 (BLE), and a Gigabit Ethernet interface. With these specifications, the RPi can easily function as a gateway in the IoT network, where multiple IoT devices constituting the WSN are able to deliver their data to the fog layer of the network using the aforementioned wireless transceivers. In addition, the RPi can be used, if needed, to process the data before delivering it to the fog layer, utilizing the edge computing concept.

F. Fog Node

The setup of this node can be divided into two parts: hardware and software.

1) Hardware

A High-Performance Computer (HPC) in our lab at HTU is used as the IoT fog node with an Intel(R) Xeon(R) Gold 6130 CPU running at 2.1 GHz and 64 GB of RAM. It is the infrastructure over which the application side of the IoT solution resides.

2) Software

The software setup depends on the desired functionality of the IoT solution. Basically, on top of an operating system, multiple software components are used to create an IoT platform. The heart of every IoT platform is the communication channel or broker and the database. Eclipse Mosquitto MQTT Broker is an open-source implementation broker for the MQTT protocol versions 5.0, 3.1.1, and 3.1. It is the broker we implement on our fog node for the purpose of application-layer communication. Also, we use Node-RED as the middleware that facilitates the broker-backend and broker-frontend connections to other parts of the solution. It is an event-driven programming tool for creating backend applications built on top of the Node.js framework. It is a very powerful tool for the development of IoT solutions as it needs an extremely small amount of code writing to achieve a lot of IoT-related tasks in the software stack with its user-friendly interface. An example depicting this simplicity is shown in Figure 4, where we set up the necessary nodes for debugging incoming and outgoing MQTT payloads as well as inserting the results into InfluxDB, which is the time-series database we deployed for storing the real-time data. Compared to conventional structured databases such as SQL, InfluxDB is schemaless and more suitable for real-time query to achieve data visualization or additional processing, data analytics, and machine learning tasks. A powerful aspect of using influxDB is the lightweight scripting language, which is Flux, that is used not only to query, but also aggregate, and manipulate the data with a very rich set of mathematical tools available with it. Moreover, Flux can be used to do this in real-time by utilizing the Tasks feature that can be deployed to schedule repeated and necessary patterns of computation on raw data to enrich it or prepare it for another phase of computing. We also set up a MySQL database, but it is used only to store information about the application in order to build the user interface (i.e., device ID, device name, etc.). InfluxDB comes with an interface that allows the user to define and configure data sources, explore the content of the buckets (i.e., databases), build dashboards with rich types of widgets, create scheduled tasks to be run on the streams of data, and set

 ISBN: 978-1-68558-054-4

threshold alert endpoints to monitor the connected IoT data. Figure 5 shows a dashboard we built to display various sensed parameters of one of our connected inverters. In the dashboard, we include two widgets per displayed value, one for graphing it and the other is a single-status widget that displays its latest state. Starting at the top from left to right are the RMS value of the AC output voltage, the DC voltage of the battery, and the DC voltage of the PV panel. Similarly, at the bottom are the output current, battery current, and the PV current. The previously mentioned components are all installed and configured on the HPC to build the IoT platform. Using the MQTT protocol, clients can either publish messages into a topic name or subscribe to a topic name to receive messages published by other clients or do both. To implement that concept, we used multiple topic names in the hierarchy form of $typeofData/Location/Gateway/Sensor. The topic names are:
$state/HTU/WiNS/Sensor-Node-Name
$command/HTU/WiNS/Sensor-Node-Name
$connected/HTU/WiNS

Topic names can be anything and can be organized into multiple levels by using the forward slash separator. The first topic is used to publish sensory data collected from each sensor node to the fog node. The second one is used to publish commands from the fog node to control the sensor nodes in the control system part of the setup. The third command is used to publish data about the connection and disconnection of the edge node to be able to track the connectivity state of the edge node.

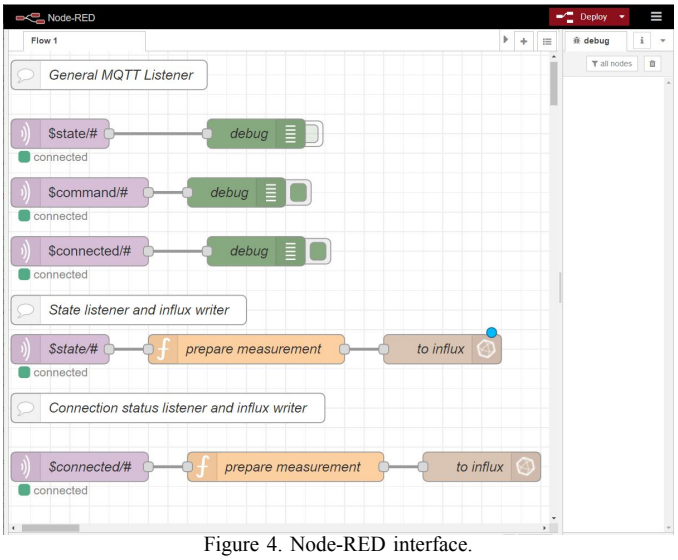


Figure 4. Node-RED interface.

IV. PERFORMANCE EVALUATION

The experimental setup, load swapping algorithm, and experimental results are presented in the following subsections. The codes used to program the WSN and implement the control algorithm in this section can be found at [17].

A. Experimental Setup

To validate the IoT system that we implemented, we used a simple algorithm for load swapping. A single source is expected to cover two different adjustable loads. The algorithm takes the sensed battery current and load current as inputs and gives the states of two switches controlling the state of each load as output. It compares the input values to predefined thresholds to determine if the connected load can be served during the targeted period. If not, it sends a command to actuate load swapping. In other words, the first load is turned off and the second is turned on. The process repeats every T seconds. We emulate the change of a load by adjusting the value of each load manually and waiting for the system to respond to the changes.

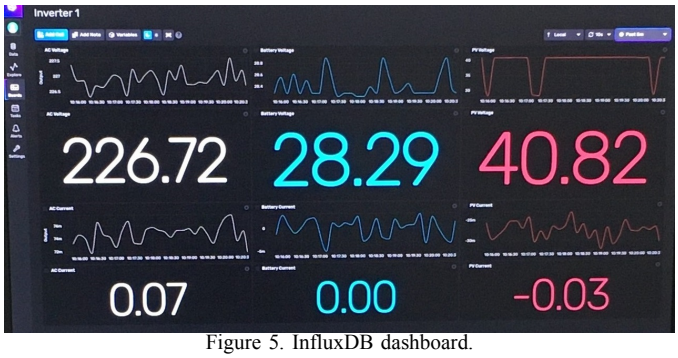


Figure 5. InfluxDB dashboard.

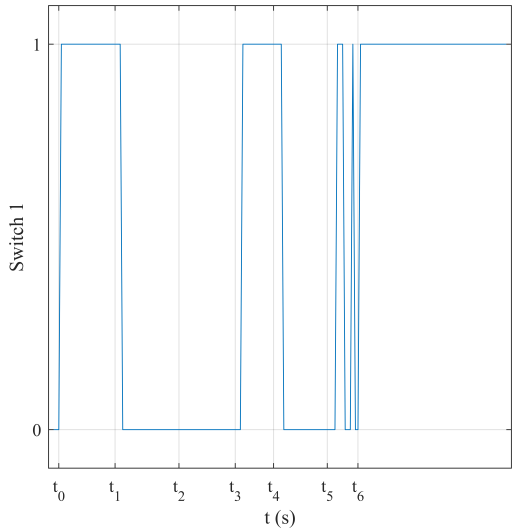
Input: I_b, I_o, $I_{\text{th},b}$, $I_{\text{th},o}$, τ_{delay}, and N.
Output: x_l, $\forall l \in \mathcal{L}$.
for $i = 1 : N$ **do**
  Set $x_0 = 1$ and unset $x_1 = 0$.
  Wait for τ_{delay}.
  Query latest I_b and I_o from database.
  while $I_b \leq I_{\text{th},b}$ **or** $I_o \leq I_{\text{th},o}$ **do**
    Wait for τ_{delay}.
    Query latest I_b and I_o from database.
  end while
  Unset $x_0 = 0$ and set $x_1 = 1$.
end for
Figure 6. Load swapping algorithm.

B. Load Swapping Algorithm

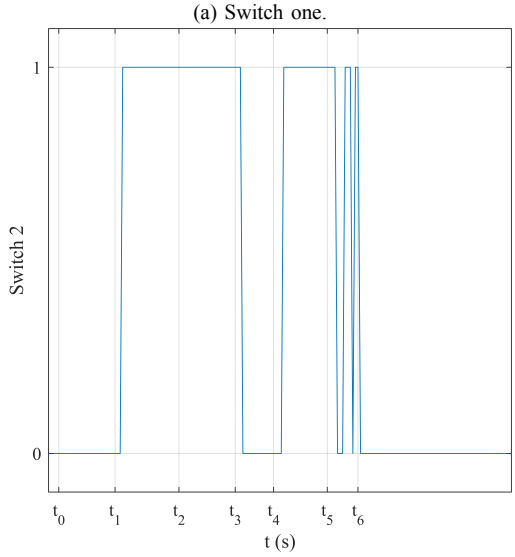
Let I_b and I_o be the battery current and the output current of the system, respectively. We define $I_{\text{th},b}$ and $I_{\text{th},o}$ as the battery current and the output current thresholds, respectively. Let $\mathcal{L} = \{1, 2, \ldots, L\}$ be the set of considered loads and let x_l be a binary decision variable. x_l equals one if load l is driven by the inverter, and it equals zero otherwise. In our case, $L = 2$ (i.e., we have two loads). Let τ_{delay} be the time difference between subsequent checks of conditions. Using the introduced notation, our load swapping algorithm for N time slots can be summarized in Algorithm 1 in Figure 6.

C. Experimental Results

We prepared our loads and ran the Python-based code that implement the algorithm and collected the following data in Figure 7 and 8. Figure 7 shows the changes in the states of load one and load two switches actuated from the fog node whereas Figure 8 captures the corresponding variations to the

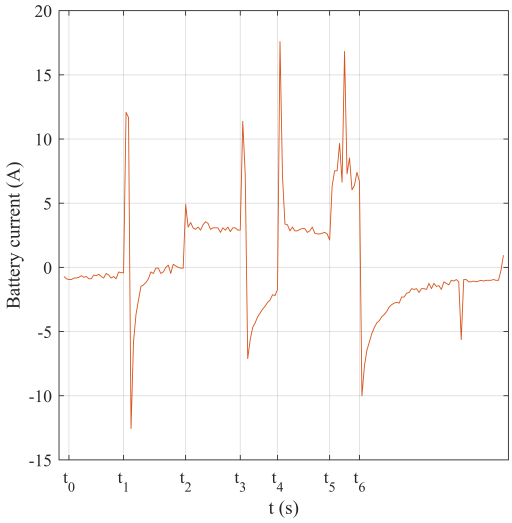


(a) Switch one.

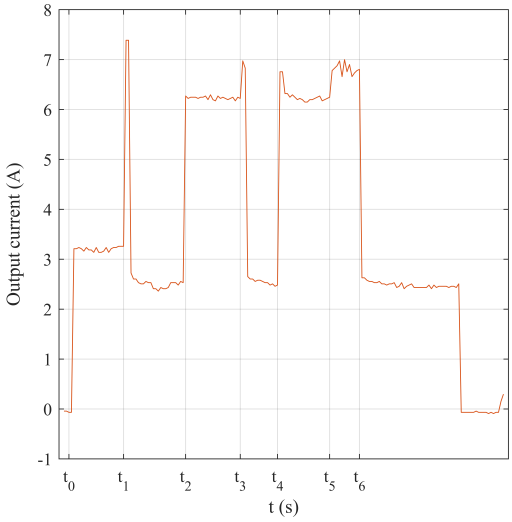


(b) Switch two.

Figure 7. The state of the switches during the testing period.



(a) Battery current (Ampere DC).



(b) Load current (Ampere RMS).

Figure 8. Current values during the testing period.

sensed battery and load currents. The scenarios start where both switch 1 and switch 2 are zero. Then, the fog node sends a $S_2S_1 = 01$ command to drive load 1 on. This action occurred at time t_0, as shown in the figures. After t_0, an increase in the load current is captured by the sensor. At the same moment, the battery current is negative which indicates that the battery is in the charging state. To push the system into using the battery we adjusted load 1 at t_1 where spikes in both the load current and the battery current are captured. This action pushed the currents beyond the threshold that we configured (5 A for each) which accordingly triggered the command that swapped the loads (i.e., $S_2S_1 = 10$). The figures confirm this and show that following this moment the switches flipped and the currents went back below the threshold. At t_2, we adjusted load 2 to be above the threshold. However, this time the battery contribution did not exceed the threshold which in turn kept everything as is in terms of the switches' states, as expected from the algorithm. Another increase of load 2 was triggered at t_3, which this time made the condition to be true again and the algorithm responded by swapping the loads as illustrated in Figure 7. t_4 shows another swap triggered by the same event. After this moment, specifically between t_5 and t_6, we fixed both loads at a heavy consumption state (the one that exceeds the thresholds). This action can be confirmed by the readings of currents during this period. The algorithm responded as expected by sending swapping commands repeatedly until the end of this period (at t_6) when we lowered load 1 to stabilize the switches eventually at $S_2S_1 = 01$.

V. CONCLUSIONS AND FUTURE WORK

We showed how each component of the IoT stack is implemented and how they are integrated with each other to collectively serve the expected purpose of the IoT, that is data collection, drawing insights, and operational improvements into the connected systems. Solving the challenge of integrating IoT into current systems will open the door for exploiting the collected data to create various algorithms and

 ISBN: 978-1-68558-054-4

optimizations in IoT-based systems. We demonstrated that by implementing a load swapping algorithm on a fog-based IoT system to control PV systems. Despite being a simple threshold-based algorithm, it served the purpose of validating the challenging IoT setup that is composed of a multitude of parts and technologies.

Other major and crucial problems in similar systems can be solved using the same setup we implemented. Examples of such problems include resource allocation, load scheduling, energy cost optimization, and energy routing. Our future work will utilize the system built during this work to perform more complicated tasks, such as energy routing based on different objectives and constraints, utilizing the true power of IoT, fog computing, and time-series databases.

ACKNOWLEDGMENT

This work was supported by the Jordan Scientific Research and Innovation Support Fund (SRISF) (grant # ICT/1/9/2018). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of Jordan SRISF.

REFERENCES

[1] A. Dawood, "Internet of things (IoT) and its applications: A survey," *Int. J. Comput. Appl.*, vol. 175, pp. 975–8887, Sep. 2020.

[2] S. H. Shah and I. Yaqoob, "A survey: Internet of things (IoT) technologies, applications and challenges," in *Proc. IEEE Smart Energy Grid Eng. (SEGE)*, 2016, pp. 381–385.

[3] B. Omoniwa, R. Hussain, M. A. Javed, S. H. Bouk, and S. A. Malik, "Fog/edge computing-based IoT (FECIoT): Architecture, applications, and research issues," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4118–4149, Jun. 2019.

[4] M. Ruiz, E. Álvarez, A. Serrano, and E. Garcia, "The convergence between wireless sensor networks and the Internet of things; challenges and perspectives: A survey," *IEEE Latin Amer. Trans.*, vol. 14, no. 10, pp. 4249–4254, Oct. 2016.

[5] N. Khalil, M. R. Abid, D. Benhaddou, and M. Gerndt, "Wireless sensors networks for Internet of things," in *Proc. IEEE Int. Conf. Intell. Sensors, Sensor Netw. and Inf. Proc. (ISSNIP)*, Apr. 2014, pp. 1–6.

[6] A. Amjad, F. Azam, M. W. Anwar, and W. H. Butt, "A systematic review on the data interoperability of application layer protocols in industrial IoT," *IEEE Access*, vol. 9, pp. 96 528–96 545, July 2021.

[7] B. M. Zakaria, A. S. Tag ElDien, and M. S. Elkholy, "Development of smart grid system," in *Proc. Int. Conf. Comput. Eng. and Syst. (ICCES)*, Dec. 2020, pp. 1–5.

[8] A. N. Pramudhita, R. A. Asmara, I. Siradjuddin, and E. Rohadi, "Internet of things integration in smart grid," in *Proc. Int. Conf. Appl. Sci. Technol. (iCAST)*, 2018, pp. 718–722.

[9] E. Esenogho, K. Djouani, and A. M. Kurien, "Integrating artificial intelligence Internet of things and 5G for next-generation smartgrid: A survey of trends challenges and prospect," *IEEE Access*, vol. 10, pp. 4794–4831, Jan, 2022.

[10] C. S. Abella *et al.*, "Autonomous energy-efficient wireless sensor network platform for home/office automation," *IEEE Sensors J.*, vol. 19, no. 9, pp. 3501–3512, May 2019.

[11] S. D. T. Kelly, N. K. Suryadevara, and S. C. Mukhopadhyay, "Towards the implementation of IoT for environmental condition monitoring in homes," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3846–3853, Oct. 2013.

[12] N. Vikram, K. Harish, M. Nihaal, R. Umesh, and S. A. A. Kumar, "A low cost home automation system using Wi-Fi based wireless sensor network incorporating Internet of things (IoT)," in *Proc. IEEE Int. Adv. Comput. Conf. (IACC)*, Jan. 2017, pp. 174–178.

[13] J. Lohokare, R. Dani, A. Rajurkar, and A. Apte, "An IoT ecosystem for the implementation of scalable wireless home automation systems at smart city level," in *Proc. IEEE TENCON Conf.*, Nov. 2017, pp. 1503–1508.

[14] M. Kumar, A. F. Minai, A. A. Khan, and S. Kumar, "IoT based energy management system for smart grid," in *Int. Conf. Adv. Comput., Commun. Mater. (ICACCM)*, Aug. 2020, pp. 121–125.

[15] I. Heđi, I. Špeh, and A. Šarabok, "IoT network protocols comparison for the purpose of IoT constrained networks," in *Proc. IEEE Int. Conv. Inf. and Commun. Tech., Electron. and Microelectronics (MIPRO)*, May 2017, pp. 501–505.

[16] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *Proc. IEEE Int. Syst. Eng. Symp. (ISSE)*, Oct. 2017, pp. 1–7.

[17] A. M. AlWaqfi, *Implementation Codes*, https://github.com/Adel-Alwaqfi/Implementation-of-a-Fully-automated-Optimized-Fog-computing-based-IoT-controlled-PV-Network.