



IoT-Based Monitoring and Control Systems for Window Energy Management: Design and Implementation

Yoon G. Kim¹(✉), Shenghui Chen¹, Timothy Dykhuis¹,
John Slagter², and Matthew Nauta²

¹ Calvin College, Grand Rapids, MI 49546, USA
ygk2@calvin.edu

² Mackinac Technology Company, Kentwood, MI 49512, USA

Abstract. This paper presents the design and implementation of an IoT-based system for window energy management. The system computes thermal transmittances of window units and solar heat gain coefficients through window units from measured sensor readings. It is critical to have energy efficient windows in homes and offices so as to conserve energy. In order to test the performance of the window units in realistic environments we created a system which can be used outdoors at any time. It is convenient for a user to control the system and access the measured data from a remote or even mobile site. The goal of this work was to design and implement a system that measures sensor data, transmits the data over a wireless communication link, distributes the data through the Internet, and stores the data in a database for analysis at any time and anywhere. The IoT is a computer network, in which anyone and anything can connect together anytime and anywhere [1]. It is realizable through sensing and communication technologies [2]. The IoT approach was adopted to achieve our goal due to the availability of sensing technology, wireless communication technologies, and standard computer networking protocols. We designed and implemented a system for calculating the thermal energy related parameters of the window units. The measured data from sensors were transmitted to a cloud server over cellular networks and the Internet. The data in the server can be accessed, stored, and displayed remotely. As long as electricity and cellular phone networks are available, the system can connect sensors to users. Extensive testing was conducted to verify the operation of the system. The testing and measurement results show that the system successfully performs the necessary operations to achieve the goal.

Keywords: Internet of Things (IoT) application ·
Window energy management · U-factor · Solar heat gain

1 Introduction

The Internet of Things (IoT) is an emerging technology that draws increasing interest in industry and also in the research community [2, 3]. The term IoT was first coined in 1999 [4] and reported by the International Telecommunication Union (ITU) in 2005

[1]. The IoT concept in the report embraces a vision of ubiquitous networks, in which anyone and anything (RFID tag or computing device with sensors, etc.) can connect together anytime (24/7) and anywhere (indoor, outdoor, or on the move). One of the definitions of IoT considers IoT as a network of interconnected objects based on standard communication protocols [5]. The terms “objects”, “things”, and “devices” are used interchangeably in the IoT articles [6]. The object can have a sensor, multiple sensors of the same type, or multiple sensors of different types. When the objects are interconnected, a sensor network is formed. In terms of sensor networks, the IoT network can be described as sensor networks connected over the Internet [6]. The IoT has industrial applications in healthcare service, the food supply chain, transportation, logistics, environmental monitoring [3], and modern manufacturing [7]. A study [8] classifies IoT solutions based on the IoT application domain. The IoT concept can be realizable through integrating enabling technologies: identification with RFID tags, sensor networks, and current communication technologies [2].

Due to the availability of sensors, current communication technologies, and standard computer networking protocols, the IoT approach was adopted to achieve our goal in this project. The main goal of the project was to design and implement a system which can measure sensor data, transmit the data over a wireless communication link, and distribute the data using standard computer networking protocols anytime and anywhere. In this case, the system reports thermal energy related parameters on window units.

It is reported that there are significant energy losses through building windows [9]. Insulated windows are being developed to conserve energy. U-factor or U-value is a measure of thermal transmittance. It represents “how much energy will be lost from a building through its windows” [10]. The lower the U-factor, the better the window insulates. A standard test method was developed to measure the U-factor using hot box methods [11]. Through the hot box method, the U-factor can be calculated as [11]

$$U_s = Q_s / [A_s \times (t_h - t_c)] \quad (1)$$

where U_s is thermal transmittance of test specimen (e.g., window), A_s is the projected area of a surrounding panel, Q_s is the time rate of heat flow through the test specimen, t_h is the temperature of the hot side of the hot box, and t_c is the cold side temperature of the hot box. The quantity Q_s is determined by

$$Q_s = Q_{heater} - Q_{wall} - Q_{box} \quad (2)$$

where Q_{heater} is the electrical power put into the heater, Q_{wall} is the heat loss through the walls surrounding the window, and Q_{box} is the heat loss through the hot box. The overall thermal resistance of a test specimen is called R_s [11] and is commonly used in the building industry and defined as

$$R_s = 1/U_s \quad (3)$$

Recently, a cost-effective retrofit window insulation system was developed using highly transparent window films to reduce heat losses [12]. The window units were evaluated by measuring the R-value (R_s) using the hot box methods. This paper explains how we extend the previous work in order to test the window units in more realistic environments emulating a building setting where the heat source is sunlight. We developed a unique type of IoT-based system to be operated with and installed in the Window Energy Management System (WEMSTM) [13] Test Box, which can be used outdoors anytime. The system is designed to measure the solar heat gain coefficient (SHGC) [14], which represents the radiant heat transmitted to an enclosure through a window. The R-values are also measured in situ using the center of glazing method [15]. The system can measure the SHGC and R-values of window units remotely by monitoring and controlling the associated modules of the system using wireless communications and computer networking protocols. The system can run 24/7 as long as it can access electricity and cellular phone networks. It is an IoT-based system as it includes sensors, embedded systems, actuators, wireless communications, and Internet cloud servers, which are the core components of IoT technology.

The rest of the paper is organized into sections as follows: In Sect. 2, the design and implementation of the proposed system are discussed. We illustrate the block diagram of the proposed system and present photos of implemented control systems for window energy management. Then, we explain the functionalities of each block and the relationship between the blocks in the diagram. Section 3 explains the embedded controller software. We focus on presenting the functionality of the application programs we developed. In Sect. 4, testing and measurement results are presented. Concluding remarks are presented in Sect. 5.

2 Design and Implementation of the Proposed System

This project required measurement of physical quantities, such as temperatures and pressures, from many sensors. The measurement data needed to be collected, then stored, and processed to evaluate the system under test.

Figure 1 shows the block diagram of the system implemented. The cellular 4G LTE router [16] connects the test box to a Message Queuing Telemetry Transport (MQTT) [17] server and a cloud database, MongoDB Atlas [18] server, through the Internet. The Command Center connects these servers remotely to access its data transmitted from the test box. The main controller of the system is the embedded system, Raspberry Pi 3 B + [19], which controls the cellular router, the Wi-Fi router, the fan control unit, and the data acquisition (DAQ) unit as shown in the figure. The detailed descriptions of the software of the embedded system are presented in Sect. 3.

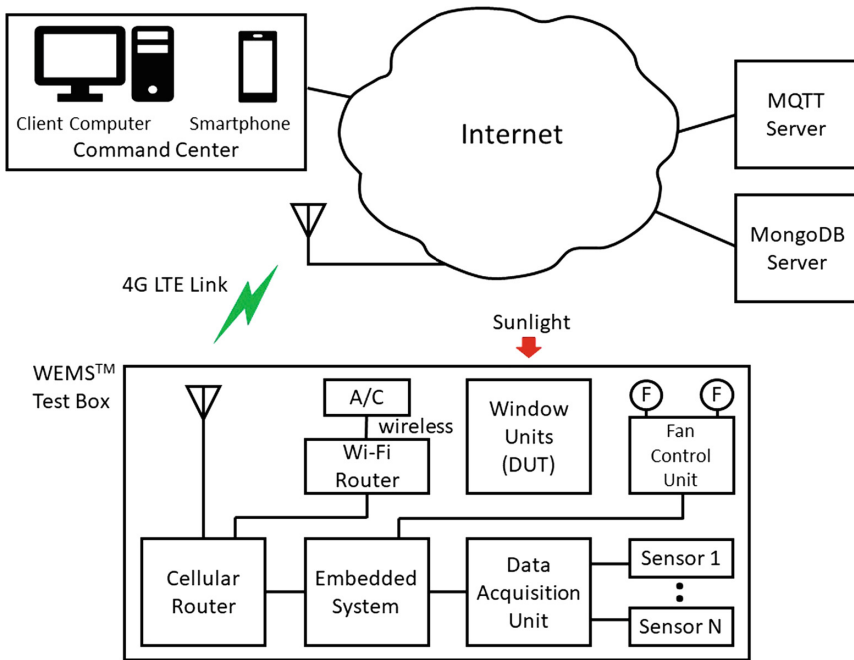


Fig. 1. Block diagram of proposed IoT-based system

The Wi-Fi router is used to connect the cellular router to the air conditioning unit, which has Wi-Fi connectivity. There are two fans inside the test box, adjusted by the fan control unit. The unit reads the current temperature or process value (PV) in the chamber where the windows units are installed and compares the setpoint value (SV). The unit adjusts the speed of the fan to minimize the difference between PV and SV in a closed-loop fashion. The SV can be configured via relay boards connected to the fan control unit by a user at the Command Center, remotely through the Internet. The DAQ unit has total of 64 single-ended input channels or 32 differential input channels [20]. Twenty four (24) thermocouples are used to measure the temperature in the chamber and on the window surfaces. Two pressure sensors [21] are used to measure the air mass flow rate at the inlet and outlet orifices. The flow rate is utilized to calculate heat transfer rate in the test box chamber interior of the window units, which allows us to compute the heat flow rate due to solar radiant heat transmitted into the chamber. The solar heat rate is utilized to determine the solar heat gain coefficient (SHGC).

All sensors are attached to the DAQ, which converts analog signals from sensors to digital data with 24-bit resolution every 1 min since the temperature being measured varies relatively slowly. The appearance of the implemented WEMSTM Test Box is shown in Fig. 2. The overall size of the system is 8 (W) × 8 (L) × 8 (H) feet excluding the roof and its air vent.



Fig. 2. Appearance of the test box in Fig. 1

Figure 3 shows the internal view of the test box (A/C, fans, and window units not shown). The DC power supply provides DC 24 V to the cellular router and the DC-to-DC converters, which provide DC 5 V and DC 9 V to low voltage circuits (such as the embedded controller, relay boards, DAQ, sensors, and Wi-Fi router).

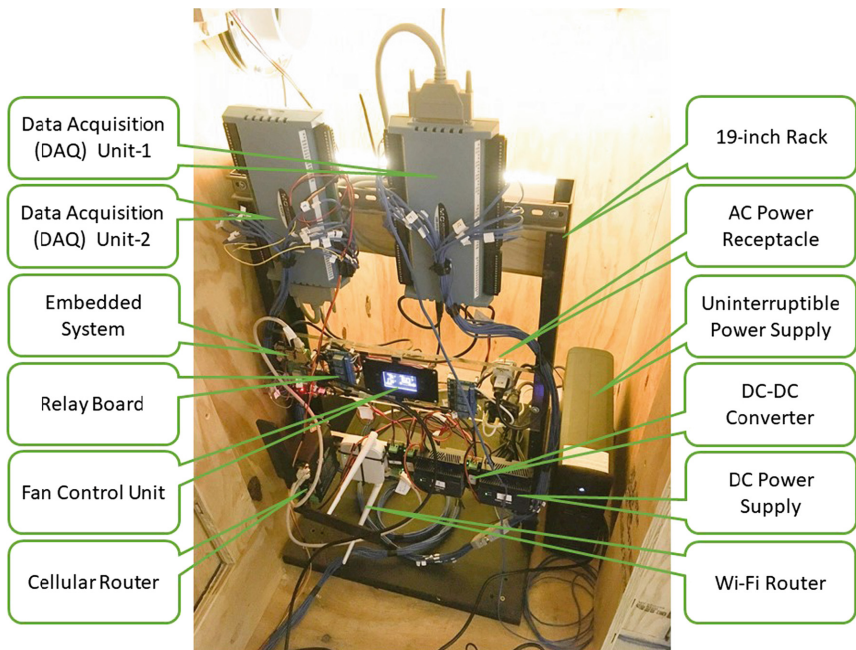


Fig. 3. Implementation of control system for the test box

The detail connection diagram is shown in Fig. 4. The red arrows represent power connections and the blue arrows represent signal connections.

The average data rate being sent from the embedded system to the cellular router should be less than the transmission throughput of the router whose memory size is finite. The measured average throughput of the router ranges between 1.3 Mbps and 7.6 Mbps, which was measured in Michigan, USA during the weekdays. The total average data rate [bits per second] can be calculated as follows.

$$\begin{aligned}
 \text{Avg. Data Rate} &= \{ \# \text{ of ch} \times (\text{ch numbers} + \text{delimiters} + \text{values}) \} \text{ samples/min} \\
 &= \{ 32 \times (6 \text{ bytes} + 5 \text{ bytes}) \} / 60 \text{ sec} \\
 &= 2,816 \text{ bits} / 60 \text{ sec} = 46.9 \text{ [bps]}
 \end{aligned}
 \tag{4}$$

As the measured throughput (1.3–7.6 Mbps) of the cellular router is practically much larger than the average data rate above, the wireless transmission through the cellular link is considered to have enough bandwidth for reliable communication. The cellular link was chosen for its large coverage and relative low cost with moderate bandwidth.

The sampled data by the DAQ are temporarily stored in the memory of embedded system and pushed to the MQTT server. We use CloudMQTT [22] for implementing the MQTT server. The data in the server can be accessed by a user remotely at the Command Center.

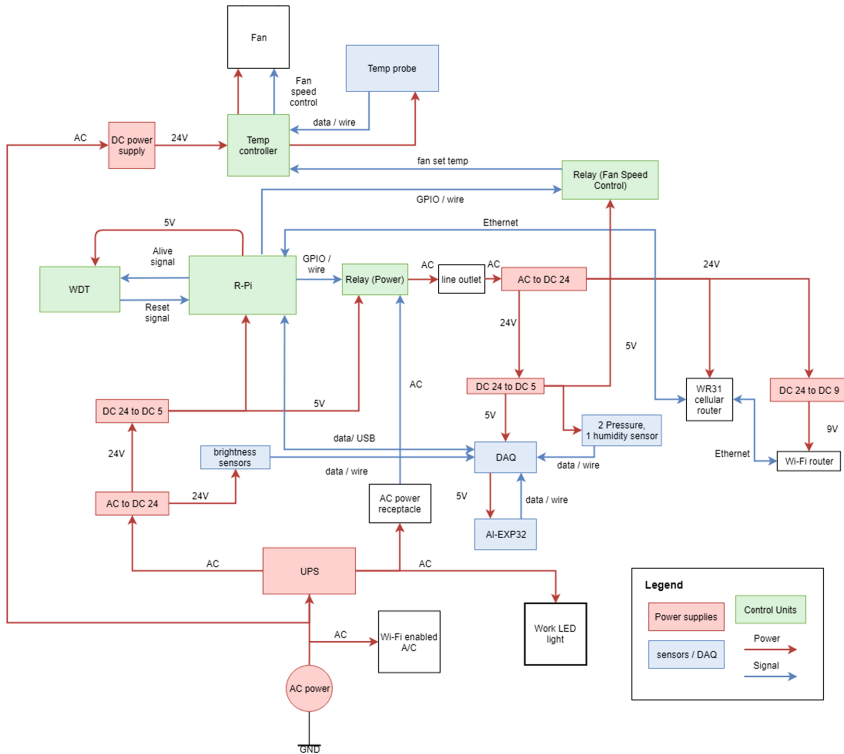


Fig. 4. Connection diagram of the control system for the test box (Color figure online)

When the system is operated remotely, automated maintenance features and user-initiated remote maintenance features are often needed to cope with situations where abnormalities occur in the test box. The automated maintenance features include (1) a watchdog timer (WDT) and (2) cellular link wake-up. The user-initiated remote maintenance features include remote rebooting and remote power cycling of routers and the DAQ. For example, when the embedded controller is halted, e.g., due to glitches on the DC power, the WDT circuit reboots the controller. This operation uses two signals connected to the embedded controller I/O pins as shown in Fig. 4. The embedded controller sends a signal (Alive) every minute. When this is received, a timer in the WDT circuit is restarted, and there is no action. If the signal (Alive) is not received for over three minutes, a signal (Reset) is sent to the controller for reboot.

The cellular router's connectivity to the Internet is monitored every thirty minutes by an embedded controller program. When the program detects errors on the Internet connection to the MQTT server, it automatically turns the Relay (Power) OFF, waits for five seconds, then back ON. This operation is to run a power cycle on the cellular router, the Wi-Fi router, and the DAQ. A user at the Command Center can also run the power cycle manually by accessing the MQTT server if needed.

3 Embedded Control Software

The embedded controller runs the control software to ensure the operation of the test box. This software includes an operating system (OS) based on Debian [23], a DAQ driver, open-source code, and the application programs we developed. This section presents the functionality of the application programs we developed, which are Watchdog_Pat, Wakeup_Post, Command_Receive, Data_Post, Internet_Check, and Test_Box.

`/etc/rc.local` file on the embedded systems is executed automatically during the boot process. The file includes Watchdog_Pat, Wakeup_Post, and Command_Receive. The last program is in an infinite loop and runs in a separate process from a main program.

`crond` (cron daemon) is the name of a daemon which runs in the background of the OS and reads a file called `crontab` (cron table) [24]. It allows shell commands to run periodically. There are three programs listed in the `crontab`: Watchdog_Pat, Data_Post, and Internet_Check.

Table 1 shows the name of programs and associated period of calls.

Table 1. Application programs and period of call

Application program	Period of call
Watchdog_Pat	1 min
Data_Post	1 min
Internet_Check	30 min

The Watchdog_Pat program is not only called by the crond every minute but also called during the boot process to prevent the watchdog timer from being timed out during the process. The program manipulates the signal on the GPIO pin of the embedded controller to refresh the WDT circuit.

The names of the MQTT topics in this project are ‘debug1’, ‘command1’, and ‘sunbox1’.

The Wakeup_Post uses a program called mosquito [25], which is installed on the embedded controller, to publish a message on the ‘debug1’ topic to the MQTT server. It establishes a connection to the server. The Wakeup_Post publishes a message to show that the embedded controller booted successfully and to provide the controller’s current IP address.

The Data_Post program creates a connection between the embedded controller and the MQTT server and then publishes messages on the ‘sunbox1’ topic. These messages are the readings of sensors attached to the DAQs. To obtain the readings, the Data_Post utilizes the DAQ device driver [26].

The Internet_Check program checks the Internet connection by attempting to connect to the MQTT server. The program uses the return value as a flag to indicate the success of the connection process. If it is not successful, the program initiates the power cycling of two routers to restart the Internet connection. If it is successful, it does nothing. This software is called every 30 min by the crond.

The Command_Receive program allows the embedded controller to receive a remote message from the MQTT server in which a user enters the message under the ‘command1’ topic on the CloudMQTT server screen as shown in Fig. 6. The program listens to a message published on a specific topic (‘command1’) and responds to four messages: ‘u’ for increasing the temperature setpoint value of the fan controller, ‘d’ for decreasing the value, ‘p’ for power cycling, and ‘r’ for restarting the embedded controller only. The program publishes a message on the ‘debug1’ topic when the connection to CloudMQTT is established successfully and when the embedded controller receives a valid command through the ‘command1’ topic. The Command_Receive program uses an open-source implementation of MQTT, paho.mqtt [27], to communicate with the MQTT server.

The Test_Box program allows the embedded controller to perform the heat flow rate calculations. This rate is integral to finding the SHGC. The program uploads the processed data to the MongoDB Atlas.

Increasing or decreasing the fan commands ‘u’ or ‘d’ are accomplished through the GPIO pins of the embedded controller. The voltages on the pins drive the relay board to close or open the pushbutton on the fan control unit shown in Fig. 3. The power cycle command ‘p’ is accomplished through the GPIO pins of the embedded controller, too. It runs a power cycle on the cellular router, the Wi-Fi router, and DAQ to restart. The restart command ‘r’ is accomplished through the restart command in the operating system.

We developed a program named mqtt_pub_sub to run on the Command Center PC. The program subscribes the ‘sunbox1’ topic so that the server can receive measurement data (messages) from the embedded controller in the test box. The program also publishes the message (‘u’, ‘d’, ‘p’, or ‘r’) on the ‘command1’ topic to display the message on the CloudMQTT screen as shown in Fig. 6.

A database is installed on the Command Center PC to have all the measurement readings from the test box stored locally on the PC. The database used for this purpose is MongoDB [27]. When new data is published on the ‘sunbox1’ topic, the mqtt_pub_sub program reads the data, parses it into corresponding fields, and stores it in the database.

The software list and devices that run the software are shown in Table 2.

Table 2. Devices and software

Device	Software
Embedded controller	Debian OS, DAQ Device Driver, paho.mqtt, mosquitto, Command_Receive, Data_Post, Wakeup_Post, Watchdog_Pat, Internet_Check, Test_Box
Command center PC	mqtt_pub_sub, MongoDB
Cloud server	CloudMQTT, MongoDB Atlas

4 Testing and Results

When the embedded controller gets booted, two messages are published on the ‘debug1’ topic as shown in the blue box of Fig. 5.

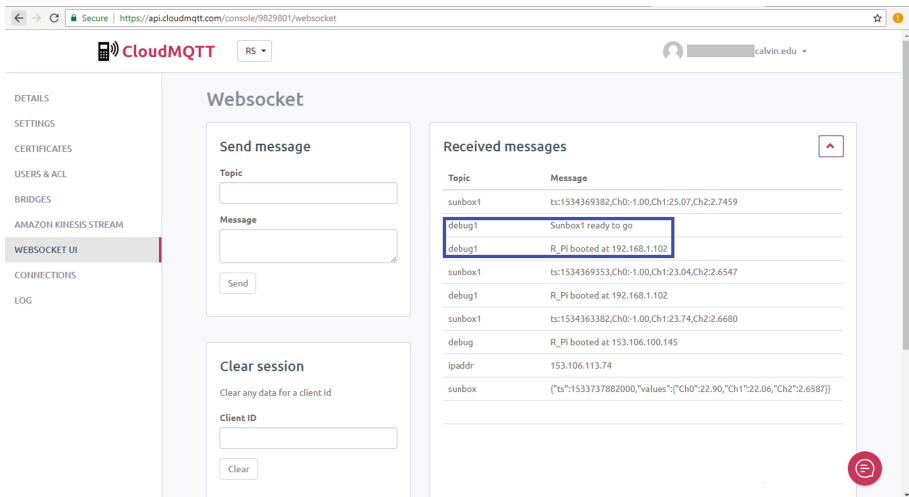


Fig. 5. Messages during boot process (Color figure online)

When a user at the Command Center types in ‘command1’ in the Topic box and ‘u’ in the Message box followed by clicking on the Send button, the message is published on the ‘command1’ topic and delivered to the embedded controller. After the Command_Receive program running on the embedded controller receives any of the four

messages listed above, the program acts according to the messages. It also publishes a message under the ‘debug1’ topic on the server acknowledging the reception of the command. This is shown in the blue box of Fig. 6.

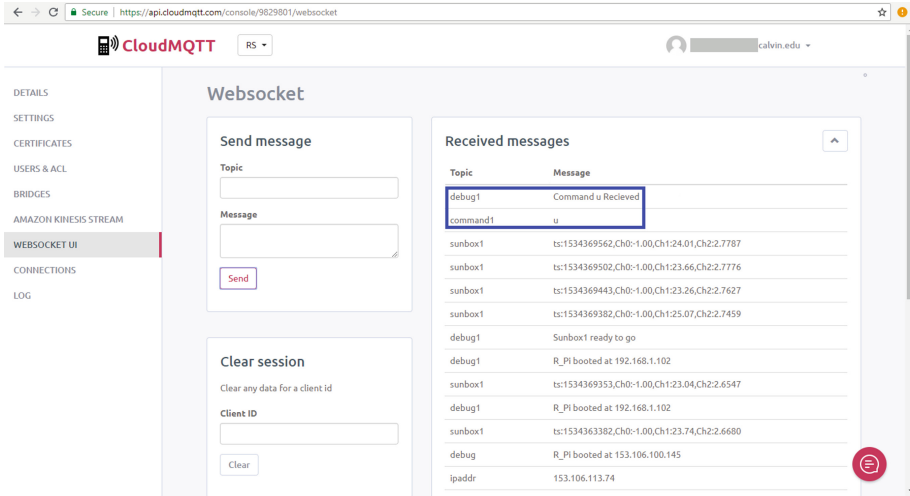


Fig. 6. Topics and messages by Command_Receive (Color figure online)

The Data_Post on the embedded controller publishes measured temperature values as messages on the ‘sunbox1’ topic, sending to the MQTT server every minute as shown in Fig. 7.

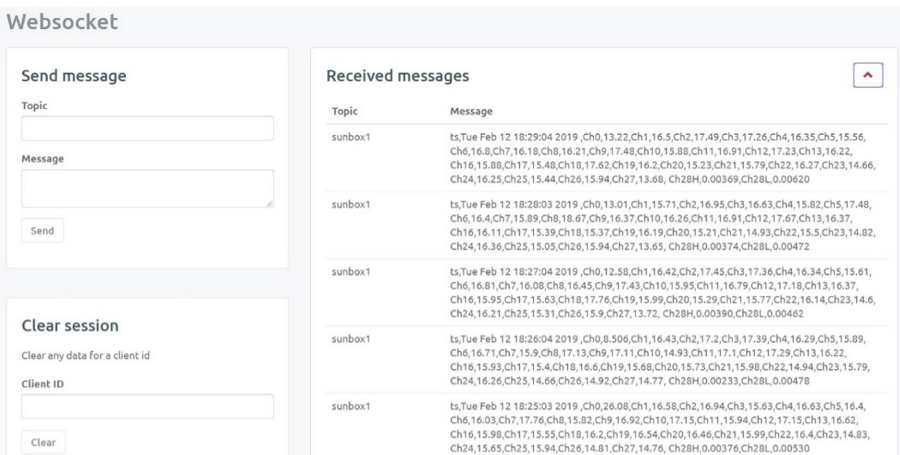


Fig. 7. Measurement data published as messages by Data_Post

The messages in Fig. 7 are in the form of a string. The string has fields that are separated by commas. The first field text “ts” stands for “time stamp”. The next field is the time stamp of the message followed including day, time, and year. The field after the time stamp is “Ch0”, indicating that the following field is the temperature in Celsius of the thermocouple connected to channel 0 of the DAQ. The same rule applies to the readings from channels 0–13 and channels 16–27. Note that the channels 14 and 15 are reserved for future use. The differential input of Channel 28 is used to measure pressures from two single-ended pressure sensors. The value following the field “Ch28H” is the pressure reading of the sensor attached to the + terminal of the channel 28 of the DAQ. Similarly, “Ch28L” refers to the pressure sensor connected to the terminal of the channel 28.

Once the Test_Box program uploads the processed data to MongoDB Atlas, charts can easily be made and customized within a MongoDB dashboard. Variables from the database are dragged to applicable fields depending on chart type for the X-axis and Y-axis values. Charts automatically update with new data, though filters may be used to select specific date ranges. Labels, colors, and other chart elements may also be customized within the chart-creation window. Figures 8, 9, and 10 illustrate the charts of time-indexed interior chamber temperature, air mass flow rate, and heat flow rate of the test box, respectively.

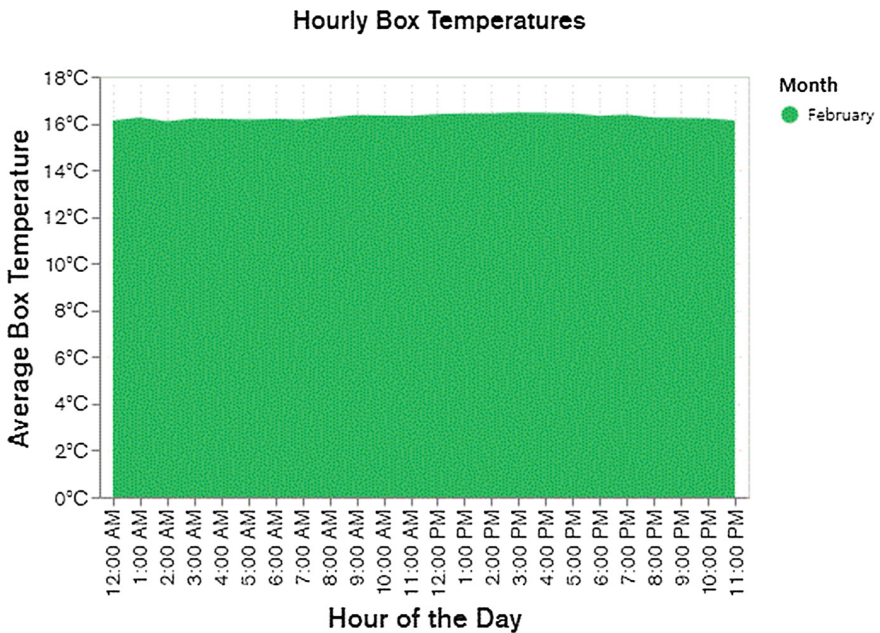


Fig. 8. Interior chamber temperature of the test box

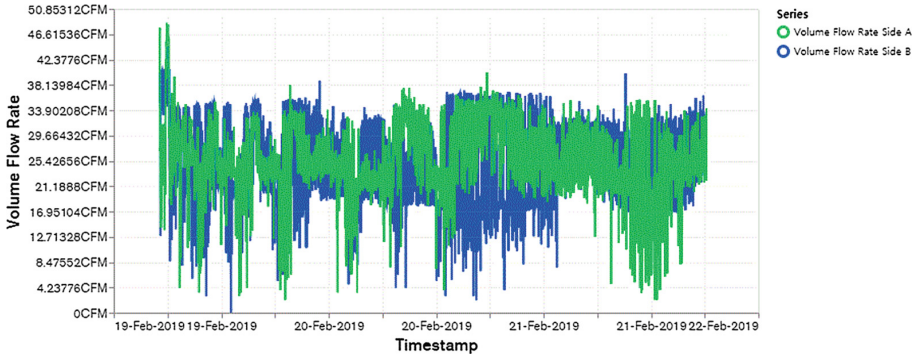


Fig. 9. Air mass flow rate of the test box

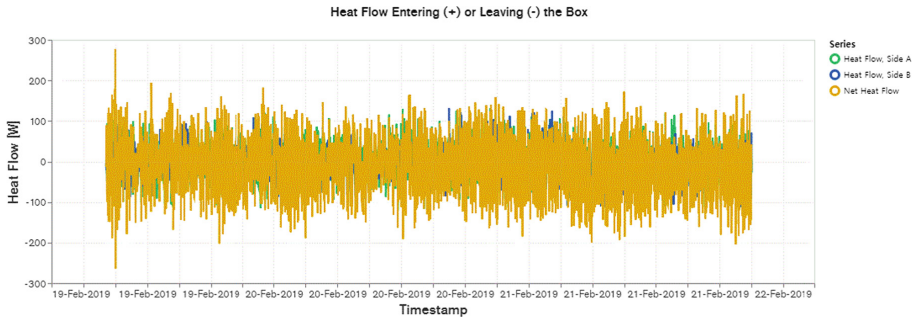


Fig. 10. Heat flow rate of the test box

5 Conclusions

This paper describes the design and implementation of an IoT-based system that measures multiple sensor data, processes the data, transmits the data over a cellular network, collects the data in a cloud server, stores the data, and displays the data remotely. It can operate anytime (24/7) and anywhere (indoor or outdoor) over the Internet and cellular network. The described system can calculate the critical values on window units, such as heat flow rate, thermal transmittance, and solar heat gain coefficient using the stored data for assessing thermal performance of the units under test.

A user can access the measured and processed data remotely from a fixed or moving site.

As the results showed, the system successfully performed the operations listed above. The goal of this project was achieved by an IoT approach with enabling technologies. The processed data were transferred to the cloud servers. The data in the database server were accessed and displayed remotely at the Command Center. Due to the automated maintenance features we developed, the system provides reliable operation.

Through the design and implementation of this project, an IoT-based system for window energy management was realized in an industrial application.

Acknowledgment. The authors thank Dr. Rich DeJong, Dr. Randy Brouwer, and Mr. Leonard Pearlman for valuable input and discussion. The IoT-based system was designed and implemented at Calvin College with funds provided by the 2018 Engineering Sustainability Research Fund. The construction of the WEMSTM Test Box and installation of the IoT-based system were conducted at Mackinac Technology Company and funded under PE 0603734A, Project T15 Military Engineering Technology Demonstration (Congressional Add), Task 07 under Contract W9132T-19-C-0001, managed by the US Army Engineer Research and Development Center, with permission granted by the Director, Construction Engineering Research Laboratory to publish this information.

References

1. International Telecommunication Union: ITU Internet Report 2005: The Internet of Things, <https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf>. Accessed 07 Jan 2019
2. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: a survey. *Comput. Netw.* **54**(15), 2787–2805 (2010)
3. Xu, L.D., et al.: Internet of Things in industries: a survey. *IEEE Trans. Industr. Inf.* **10**(4), 2233–2243 (2014)
4. Ashton, K.: That ‘Internet of things’ Thing. <http://www.rfidjournal.com/articles/view?4986>. Accessed 07 Jan 2019
5. European Commission: Internet of Things in 2020. https://docbox.etsi.org/erm/Open/CERP-IoT20090518/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v3.pdf. Accessed 07 Jan 2019
6. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Context aware computing for the Internet of Things: a survey. *IEEE Commun. Surv. Tutorials* **16**(1), 414–454 (2014)
7. Bi, Z., Xu, L.D.: Internet of Things for enterprise systems of modern manufacturing. *IEEE Trans. Industr. Inf.* **10**(2), 1537–1546 (2014)
8. Perera, C., Liu, C.H., Jayawardena, S.: The emerging Internet of Things marketplace from an industrial perspective: a survey. *IEEE Trans. Emerg. Top. Comput.* **3**(4), 585–598 (2015)
9. U.S. Department of Energy: Update or Replace Windows. <https://www.energy.gov/energysaver/design/windows-doors-and-skylights/update-or-replace-windows>. Accessed 07 Jan 2019
10. NFRC: Understanding Window U-Factor. <https://www.nfrccommunity.org/blogpost/925129/226115/Understanding-Window-U-Factor>. Accessed 07 Jan 2019
11. ASTM: Standard Test Method for Measuring the Steady-State Thermal Transmittance of Fenestration Systems Using Hot Box Methods. https://compass.astm.org/EDIT/html_annot.cgi?C1199+14. Accessed 07 Jan 2019
12. ARPA-E: Single Pane Window Retrofit System. <https://arpa-e.energy.gov/?q=slick-sheet-project/single-pane-window-retrofit-system>. Accessed 07 Jan 2019
13. Mackinac Technology Company: Mackinac WEMS. <https://www.mackinac-technology.com/our-product.html>. Accessed 07 Jan 2019
14. U.S. Department of Energy: Energy Performance Ratings for Windows, Doors, and Skylights, <https://www.energy.gov/energysaver/design/windows-doors-and-skylights/energy-performance-ratings-windows-doors-and>. Accessed 07 Jan 2019

15. NFRC.NFRC 100-2017 [E0A2]: <https://www.nfrcommunity.org/store/ViewProduct.aspx?id=1403208>. Accessed 07 Jan 2019
16. DIGI. TransPort WR31: <https://www.digi.com/products/models/wr31-m52a-de1-tb>. Accessed 07 Jan 2019
17. MQTT.org: MQTT. <http://mqtt.org/>. Accessed 07 Jan 2019
18. MongoDB: MongoDB. <https://www.mongodb.com/>. Accessed 07 Jan 2019
19. Raspberrypi.org: Raspberry Pi. <https://www.raspberrypi.org/>. Accessed 07 Jan 2019
20. Measurement Computing: USB-2416 Series. <https://www.mccdaq.com/usb-data-acquisition/USB-2416-Series.aspx>. Accessed 07 Jan 2019
21. Honeywell: Board Mount Pressure Sensors. https://sensing.honeywell.com/index.php?ci_id=151134. Accessed 07 Jan 2019
22. Cloudmqtt: CloudMQTT. <https://www.cloudmqtt.com/>. Accessed 07 Jan 2019
23. Debian: Debian OS. <https://www.debian.org/>. Accessed 07 Jan 2019
24. Opensource: How to use cron in Linux, <https://opensource.com/article/17/11/how-use-cron-linux>. Accessed 07 Jan 2019
25. Mosquitto.org: Eclipse Mosquitto. <https://mosquitto.org/>. Accessed 07 Jan 2019
26. Measurement Computing: Third-Party Linux Support. <https://www.mccdaq.com/daq-software/Linux-Support.aspx>. Accessed 07 Jan 2019
27. Eclipse.org: Eclipse Paho. <https://www.eclipse.org/paho/>. Accessed 07 Jan 2019