Deploying IoT Devices to Make Buildings Smart: Performance Evaluation and Deployment Experience

Xiangyu Zhang, Rajendra Adhikari, Manisa Pipattanasomporn, Murat Kuzlu, and Saifur Rahman
Bradley Department of Electrical and Computer Engineering and Advanced Research Institute,
Virginia Tech, Arlington, VA, USA 22203

***Abstract*— This paper summarizes the authors' work of deploying Internet of Things (IoT) to build a reliable, cost effective and versatile energy management system for small- and medium-sized commercial buildings. This paper addresses critical issues related to the choices and evaluation of embedded systems and software optimization to enable the deployment on a low-cost single board computer, and experience of integrating IoT devices to enable smart building operation. It then discusses three demonstration projects located in Virginia and summarizes deployment experience.**

***Index Terms*—IoT devices, Embedded System, Smart Building**

I. Introduction

Internet of Things refers to the concept of making devices capable of connecting to the Internet, enabling them to work together to best serve the user. The number of Internet-capable devices is growing and is projected to reach 6.4 billion by the end of 2016 [1]. As a result, the proliferation of IoT enabled scientific research has also been observed. For example, research on IoT for smart cities is discussed in [2]. Authors in [3] propose a multi-layer vehicular data cloud services to solve transportation issues. Authors in [4] propose an IoT-based method for manufacturing resources intelligent perception and access in cloud manufacturing. As information security is also crucially important in any IoT-based project, authors in [5] give an overview on IoT security issues; and a lightweight key establishment protocol suitable for home environment is proposed in [6].

In residential and commercial buildings, IoT devices offer opportunity to save energy by providing users with a tool to remotely monitor and control, set schedules and provide real time feedback on energy usage. A domestic condition monitoring system based on low cost ubiquitous sensing system is proposed in [7]. Authors in [8] demonstrate a HVAC control strategy using IoT devices to minimize energy usage considering occupant thermal comfort. A home energy management system is proposed in [9] that can distinguish occupant activities and control devices intelligently.

While most previous work focuses on theoretical or conceptual frameworks, this paper demonstrates the full workflow of deploying IoT devices to enable smart building operation. A brief introduction to the open source software for building energy management deployed with IoT devices is presented in Section II. Evaluation of embedded system that hosts such software is discussed in Section III. Section IV discusses IoT device integration. Section V discusses approaches for improving software performance on a single board computer, and Section VI discusses the deployment experience in three buildings.

This work was supported in part by the U.S. Department of Energy under Contract DE-EE 0006352.

II. Building Energy Management Open Source Software (BEMOSS)

Funded by the U.S. Department of Energy, Virginia Tech has developed a Building Energy Management Open Source Software (BEMOSS) platform, which is an open source solution to help buildings operate more efficiently and save energy [10]. The software platform targets small- and medium-sized commercial buildings, up to 50,000 square feet. These buildings constitute almost 95% of the total number of buildings in the United States, and they do not typically have existing building automation systems. The current release of BEMOSS is Version 2.0 and is available on Github [11].

Built upon VOLTTRON™, a platform developed by the Pacific Northwest National Laboratory, BEMOSS also follows the multi-agent structure. BEMOSS agents comprise both system-level agents (e.g., a platform agent, a device discovery agent, a multi-node agent, etc.) and device agents (e.g., thermostat agents, lighting load agents and plug load agents). Following features are offered:

- *Plug & Play:* Using a dedicated device discovery agent, one of the BEMOSS key features is its ability to automatically discover supported devices in buildings;
- *Interoperability:* BEMOSS is capable of communicating with IoT devices that use different communication technologies and data exchange protocols; these devices can be from different manufacturers.
- *Ability for remote control and monitor:* BEMOSS allows users to monitor and control supported devices in real time and remotely via its built-in web server.
- *Cost Effectiveness:* BEMOSS is designed to be light-weight to operate on a low-cost single board computer.
- *Open Source, Open Architecture:* BEMOSS is designed to have an open architecture to make it easy for hardware manufacturers to seamlessly interface their IoT devices with BEMOSS. Moreover, it is also designed to allow software developers to easily contribute to the platform by adding additional devices, functionalities and applications.

978-1-5090-4130-5/16/$31.00 ©2016 IEEE

Some of these features require considerable computation resources, and thus might increase the hardware investment. To enable the above merits while keeping the budget low, the following two aspects need to be considered when designing and deploying such a system: (i) a host machine/cloud service should have excellent performance at a low cost; and (ii) from the software perspective, BEMOSS should be adaptive to host machines. These issues are discussed in Sections III and V.

III. An Embedded System as a BEMOSS Host

Though cloud computing has become widely used, the service fee is still costly. For example, the t2.small service by Amazon Web Service EC2, which is equivalent to one virtual CPU and 2GB memory, costs more than $200 annually for its Linux usage charge as $0.026 per hour. Thus, to enable a cost-effective building energy management solution, BEMOSS is designed to run on an embedded system. As of 2016, the number of choices of such an embedded system are dramatically increasing. To find out the most suitable embedded system for BEMOSS, three popular single board computers (SBC) are selected to compare for their performance when running BEMOSS. These SBCs are listed in Table I.

TABLE I. Single Board Computers for Testing

SBC	Processor	Memory	Price
Odroid-XU4	CPU: ARM Cortex A15×4 @2GHz, A7×4 @1.4GHz GPU: ARM Mali-T628 MP6	2GB	$ 74.00
Cubieboard4 CC-A80	CPU: ARM Cortex A15×4 @2GHz, A7×4@1.3GHz GPU: PowerVR G6230	2GB	$ 138.00
Wandboard-Quad	ARM Cortex A9×4@1GHz GPU: Vivante GC 2000 + GC 355 + GC 320	2GB	$ 129.00

Both Cubieboard and Wandboard using SD cards to host the operating system while Odroid uses the eMMC. To test which SBC is best for BEMOSS operation, a performance test that involves the following four test cases are designed:

a) Case 1: Only Linux OS and Virtual Network Computing (VNC) server (for remote control) are running.

b) Case 2: Linux OS, VNC and BEMOSS (with the webserver and databases running, but no device to monitor and control). This case has 9 system agents.

c) Case 3: Linux OS, VNC and BEMOSS that monitors/ controls five devices. BEMOSS has the total of 14 agents running, including nine system agents and five device agents.

d) Case 4: Linux OS, VNC and BEMOSS that monitors/ controls 20 devices. BEMOSS has the total of 29 agents running, including nine system agents and 20 device agents.

To compare the SBCs' loading ability, the Linux command 'top' is used to check the load average in the past 1 minute, 5 minutes and 15 minutes. Because the load average in 1 minute can be volatile, the 15-minute average data are used here for comparison. Table II summarizes this load average test result.

The percentage is acquired using measured value divided by the board's ability to process which can be represented by its core number [12]. Though Odroid-XU4 and Cubieboard4 have similar specs, Cubieboard4 shows higher load average than Odroid for the same amount of work as shown in Table II.

TABLE II. Load Average Test Result

Test Result (Linux Load Average)				
	Case 1	**Case 2**	**Case 3**	**Case 4**
Odroid	0.05	0.28	0.6	2.77
Cubieboard	1.24	1.59	2.61	7.67
Wandboard	0.21	0.40	1.58	4.46
Test Result (Loading Percentage)				
	Case 1	**Case 2**	**Case 3**	**Case 4**
Odroid	0.63%	3.50%	7.50%	34.63%
Cubieboard	15.50%	19.88%	32.63%	95.88%
Wandboard	5.25%	10.00%	39.50%	111.50%

According to Phoronix Test Suite (PTS), performance of Odroid-XU4 and Cubieboard4 is presented in Table III, higher values in both CPU/RAM categories imply better performance. This explains the performance difference in Table II. Therefore, Odroid-XU4 appears to be the most powerful SBC for hosting BEMOSS. However, it is worthwhile to point out that this comparison is based on the scenario of running BEMOSS and cannot be generalized to other use cases.

TABLE III. PTS Benchmark Test Result

	Unit	**Odroid-XU4**	**Cubieboard4 CC-A80**
CPU (Multi-core)	MFLOPS*	171	158
RAM - Floating Point Average	MB/s	4,732	3,260

**** Mega floating-point operations per second***

Though Odroid has very good performance running BEMOSS, the big RAM consumption hinders it form running more agents. Figure 1 shows the RAM consumption in four cases. In Case 4, where 29 agents are running in total, less than 300 MB RAM out of 2GB is available, this raises the possibility of Cassandra database failure due to RAM shortage.

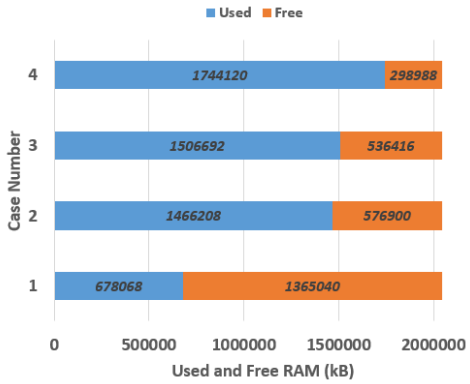

Fig. 1. Odroid RAM Usage under 4 cases

In our experience, it is a good practice to limit the number of devices that BEMOSS monitors/controls (i.e., the number of agents in operation) so that a free RAM of at least 400 MB is guaranteed. Note that the BEMOSS multi-node structure is available for accommodating more devices and enabling system scalability. This will be discussed in Section V.

IV. Supported IoT Devices

BEMOSS supports a variety of IoT devices, such as smart thermostats, lighting controllers, plug load controllers, sensors and more. Some examples of supported IoT devices and how they are integrated to BEMOSS are presented in this chapter.

A. Smart Thermostats

Smart thermostats are becoming increasingly popular, and this indicates growing adoption of the smart building idea. Four different types of Wi-Fi thermostats: Nest, Honeywell, ICM and Radio Thermostats, have been tested in the lab. While Radio thermostats support direct communication to the device with their open API, other thermostats exclusively require communication via their cloud server, which means Internet connection is required. To work with such smart thermostats, the following features are added to BEMOSS:

1) Abstraction using API Interface

All of the thermostats have the basic, heat set point, cool set point and the current temperature variables. However, the actual name of these variables as used on their screen or their app are different. Therefore, in order to provide seamless and uniform experience to the user, BEMOSS provides its API translators that allow translating different commands used by different thermostats to be commonly understood by BEMOSS.

2) BEMOSS AUTO mode

Since not all thermostats have a built-in AUTO mode, BEMOSS provides its own AUTO mode. This AUTO mode is necessary during shoulder seasons when day and night temperatures vary significantly. In an AUTO mode, a user specifies two set points, heat set point and cool set point, and the thermostat will change the HVAC mode by itself to ensure that the indoor temperature remains within the specified set points. BEMOSS own AUTO mode works as follows:

At every 20 seconds,

- Get heat and cool set points from the user (through the UI) and save them as local variables
- Compare the current temperature with those set points
- If the current temperature is less than the heat set point then change the thermostat mode to heating, and set the set point to the heat set point
- Else if, the current temperature is greater than the cool set point, change the thermostat mode to cooling, and set the set point equal to the cool set point
- Else hold on to the current mode and set point.

3) Schedule Synchronization

Different thermostats have different interfaces for handling schedules. However, except for Nest, all thermostats support the basic four-period-per-day schedule, i.e., awake, leave, return and sleep. Even though, a more complex schedule could be implemented by bringing the schedule implementation into the BEMOSS software side, it was decided to let the device itself handle the schedule by synching the schedule set by the user on BEMOSS UI into the device settings. This results in lower computational requirement for BEMOSS. Also, in case of SBC failure, the device can still follow the schedule.

4) Anti-Tampering

Another useful add-on feature implemented for thermostats, is the anti-tampering. In BEMOSS, an option of enabling or disabling anti-tampering mode for each thermostat is provided. When the anti-tampering mode is enabled, when somebody changes settings on the device (by pressing the physical buttons on the device, or through the thermostat app), BEMOSS will detect the change, notify the administrator via email or text, and change the settings back to its original value. There is also an option to provide some allowance, default of plus or minus 2 degree Fahrenheit, so that any changes within that allowance will not trigger anti-tampering. In case a change more than the allowance is made, the set point will be brought back to the original value plus or minus the allowance.

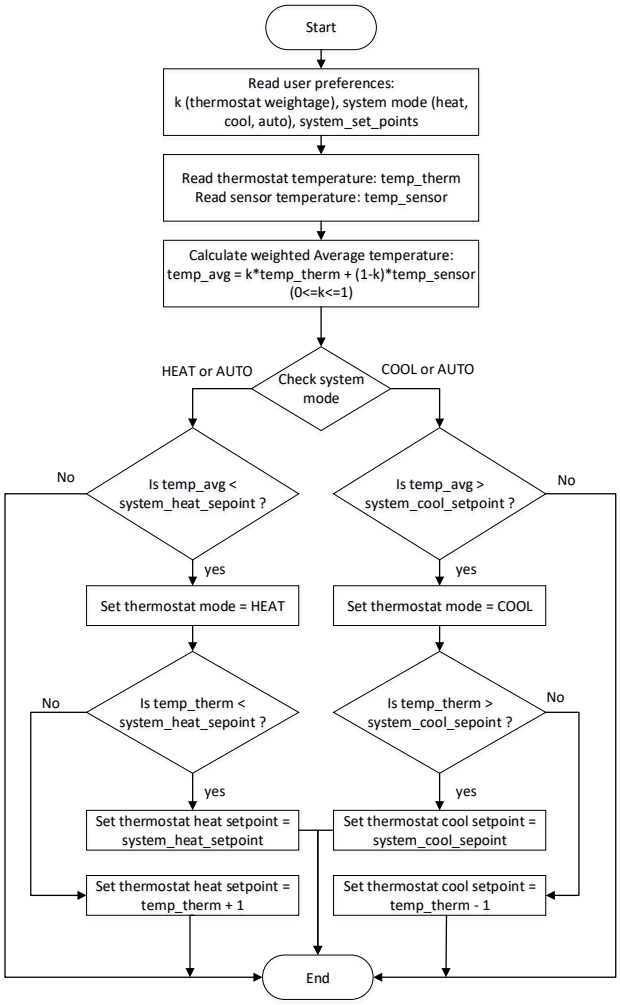

Fig. 2. Flow chart of thermostat control with an external temperature sensor

5) Integrate a temperature sensor for control of large space

In some cases, a large space may require an extra temperature sensor to provide even indoor temperature. Since most smart thermostats available today do not support an external temperature sensor, BEMOSS provides the feature to pair a temperature sensor with a smart thermostat. A user defined weight will be used to calculate the room temperature for the control. Figure 2 shows the logic for such control.

B. Lightings

BEMOSS has been integrated with IoT-enabled lighting devices, such as Particle Photon-driven step-dim fluorescent lighting and Philips Hue, as discussed below.

1) Particle Photon-driven step dim fluorescent lighting.

The Particle Core/Photon is a Wi-Fi enabled IoT chip that provides flexible API for developers to customize. In BEMOSS, Photon has been used to control a step-dim ballast, which drives two relays to control fluorescent lighting. It allows the brightness to change from 0%, 50% to 100%. See Table IV. This showcases a means to integrate legacy devices into BEMOSS. The use case is provided below.

TABLE IV. STEP DIM BALLAST CONTROL

Two Switches Condition	Power
(0, 0)	0%
(0, 1) or (1, 0)	50%
(1, 1)	100%

a) Use Case:

In a classroom with large windows, when nature light is sufficient, indoor lights can be dimmed to 50% to save energy.

b) Hardware:

Four IO pins of Photon are used to drive four relays of Particle relay shield. These four relays are used to control two fixtures of florescent lights, two for each.

c) Software – Customizing Particle App:

Particle provides an open API builder (build.particle.io) which enable developers to customize the app runs on Photon. In order to discover/control/monitor lighting devices for BEMOSS, a self-defined app with six cloud variables and three functions are developed. See Table V.

TABLE V. PARTICLE APP FOR STEP DIM BALLAST CONTROL

Variables	**Description**	
DZERO	Value of 'D0' pin, 0 for low and 1 for high	Value can be changed by setVariable function.
DONE	Value of 'D1' pin, 0 for low and 1 for high	
DTWO	Value of 'D2' pin, 0 for low and 1 for high	
DTHREE	Value of 'D3' pin, 0 for low and 1 for high	
ipString	IP Address of this Particle device	
macString	MAC Address of this Particle device	
Functions	**Description**	
setup	Initialize variables, acquire IP/MAC address.	Execute when start
setVariable	Used to set value to variables, such as DZERO	Called by API function 'setResult'
digitalWrite	Change the output of Particle device output pins	Called by API function 'digitalwrite'

Variables 'ipString' and 'macString' are used for BEMOSS to identify which Photon chip to control and the rest of the cloud variables represents the status of four IO pins.

Function 'digitalWrite' is responsible for setting IO pin status; 'setVariable' is used to change the status of cloud variable, namely record the IO pin status in the cloud. This allows BEMOSS to know the device status at any time. App can be flashed to the chip through the Particle App portal.

For BEMOSS (python-based) to communicate with the device, an API interface is developed to allow discovery, control and monitor of the Photon-based lighting system. Such API is modified based on Alidron/spyrk from GitHub [13]. This enables Photon discovery and allows BEMOSS to call all self-defined functions mentioned above.

2) Philips Hue.

Philips Hue is a set of LED light bulbs with a bridge that allows a user to control brightness and color wirelessly. Both 2012 and 2015 versions of Philips Hue hubs have been integrated to BEMOSS. With the help of the RESTful API, BEMOSS can easily check the status of the device, turn ON/OFF the device, change the brightness, set lighting operating schedule, change the bulb color and retrieve historical usage information through BEMOSS UI.

C. Other IoT devices

BEMOSS can also be integrated with a customized wireless temperature sensor developed using Raspberry Pi and a one-wire digital temperature sensor [14]. The temperature sensor would be connected to one of the Raspberry Pi GPIO pin and the Pi would communicate using one-wire protocol to get the temperature data. See Figure 4.

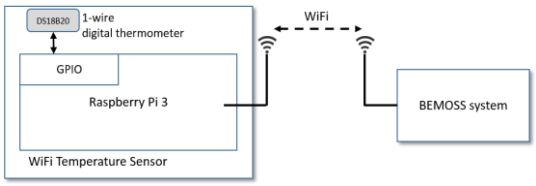

Fig. 3. *Temperature sensor made from Raspberry Pi*

This Pi and the sensor together can act as a Wi-Fi temperature sensor that can be placed at a desired location. Whenever BEMOSS needs to get sensor readings, it can simply communicate with the Pi through a Wi-Fi network. This can be achieved by running a primitive web-server on the Pi, which can respond to a temperature reading query on its TCP/IP address. The capacity to support these kinds of customized IoT devices can certainly help bring the deployment cost down.

V. BEMOSS PERFORMANCE IMPROVEMENT

This section discusses three techniques implemented to make BEMOSS suitable for deployment on SBCs.

A. Data Saving Technique

Considering the limited storage on Odroid, a data filtering technique is implemented to save disk space by reducing the number of unnecessary data entries. Taking advantage of the NoSQL database, BEMOSS only saves data points that are different from their previous stages. This results in a considerable disk space saving. Take the example below, where data for a thermostat are saved only when they change. Assuming that change rates are:

–Temperature change: every 3 minutes
–Battery level change: every 10 minutes
–Set point change: every 2 hours
–Thermostat mode change: every week

With these rate and with data filtering implemented, 0.74MB of disk space will be consumed per month for a single device. On the other hand if no data-filtering is employed it will take 13.4MB per month. This indicates the savings of 95% disk space in a month.

In addition, runtime experiments are also conducted: three instances of BEMOSS is running with different number and types of smart devices, the average data increment speed (KB/hr) is summarized in Table VI. It is worthwhile to point out that the data increment speed varies according to the types and number of IoT device monitored and controlled by BEMOSS. Different devices record different number of data points at different intervals. According to Table VI, none of these three scenarios demonstrates an extremely high data generating speed. Thus, this shows that after the implementation of such filtering mechanism, the problem of storage limit in Odroid is mitigated. That is, it will take between 3,927-173,032 days (or 10-474 years) to fill 20 GB out of a 32 GB eMMC when BEMOSS monitors and controls 6-15 devices simultaneously.

TABLE VI. DATA INCREMENT SPEED

BEMOSS Instance	**1**	**2**	**3**
Number of devices	15	6	13
Average data increment (KB/hr)	159.60	5.05	222.50
Days to fill up 20 GB:	5475	173032	3927

In BEMOSS, a custom function is used to detect the change, because variables like the power in power meter, or the light illuminance in ambient light sensor can change slightly (less than 0.5%) every device monitoring time because of the noise in the system. Hence, a tolerance is defined and any changes below the tolerance are discarded as irrelevant. Also as a fallback procedure, the data is saved in database at every 15 minutes even if nothing is changed.

B. Agents Hibernation

In order to support BEMOSS plug and play functionality, an agent exists on BEMOSS to broadcast discovery messages on the network, or ping a range of IP addresses. However, after the initial surge of discovery, the process is repeated with increasingly less frequency, so that no valuable computational power is wasted for always trying to discover devices. Instead, manual discovery is employed. That is, once the initial discovery process is finished, and if the user needs to immediately discover some newly added device, a discovery command can be issued to look for the particular kind of device added. This is instead of scanning for the whole range of supported devices all the time.

In addition, the data filtering mechanism mentioned above also relieves all device agents from doing unnecessary logging and thus reducing the Odroid's burden.

C. Multi-node Structure

Even with various measures taken to reduce resource consumption there is a limit on how many devices can be supported by a single SBC, as pointed out in Section III. In order to cope with scenarios where the number of IoT devices exceeding the limit of a single SBC, a robust master-slave multi-node architecture is introduced to facilitate this scalability issue. The master node hosts the webserver and is responsible for keeping track of slave nodes. Any device agents can be migrated between various nodes to properly balance the resource consumption among them. Since the core and nodes all reside on the same network, the information about the device access location (IP address and ports) can be freely shared between them (via encrypted communication among them). The benefit is that the discovery process do not need start again when device agents are migrated. Also, any authentication credentials to access the devices can be shared. As for saving time series data, while it is possible to have only one database server running on the master, and have all agents of the slave nodes connect to that server, it is not the best solution in terms of reliability and speed. Instead, Cassandra distributed database system is used, with each node hosting one instance (node) of the database server, and each device agent connecting to the database server on its own node. Due to the transparently distributed nature of the database, and the option of having data replication, any data saved by device agent on any node, is always available everywhere. The web server on the master node can connect to the database server on the master node, and still be able to show the chart of historical usage of various device agents. Also, having data replication ensures that even if one of the node is damaged or becomes out-of-service, none of the data is lost.

VI. DEPLOYMENT AND OPERATION

After Odroid XU4 has been installed with BEMOSS, it can be taken to the building for deployment. Currently, the BEMOSS platform has been deployed in three buildings located in Alexandria, Arlington and Blacksburg, Virginia with the longest operation time of more than one year.

A. Consideration for on-site Deployment

1) Security:

BEMOSS requires password authentication to log in, and only the 'Admin' and 'Zone Manager' roles can conduct corresponding control. All control activities can be logged for future reference. Physically, the Odroid is locked in a NEMA box in an electrical room with minimum access to tenants.

2) Maintenance:

Remote access is enabled for maintenance such as BEMOSS code update to eliminate the need to visit the building. Depending on the type of the building network, port forwarding and VPN might be needed. Little maintenance is needed for Odroid. As shown in Table VI, storage is abundant, thus eliminating the need to access Odroid physically.

3) Reliability:

Thanks to the multi-node structure that allows agents to be allocated among core and nodes, this adds an extra reliability by allowing the BEMOSS core to take over the monitoring and control of smart device operation when a node fails. In addition, BEMOSS is designed to cope with situations such as power outage contingency. When a building recovers from a power outage, Odroid will automatically boot up and start

BEMOSS to resume its job with zero human interference. Similarly, a watchdog can be implemented in case of occasional system failure.

B. Benefits of Integrating IoT Devices to Enable Smart Building Operation

By using BEMOSS that enables integration with IoT devices, building operators can enjoy:

- ***Accessibility***: Remotely control any connected device in a building from anywhere using smart phone, tablet, laptop, etc. Figure 4 shows BEMOSS dashboard that display status of all devices ;

Fig. 4. BEMOSS Dashboard

- ***Automation:*** Set schedule and provide automation of device operations in buildings. A good example is the BEMOSS ability to respond to the OpenADR signal from electricity utility, and manage load shedding automatically;
- ***Visualization:*** Provide access to historical data in graphical formats. This enables the easy-to-analyze display of device operational data. Figure 5 shows an example of the power consumption heat map of rooftop units (RTUs) in one of the demonstration building a period of about five weeks.

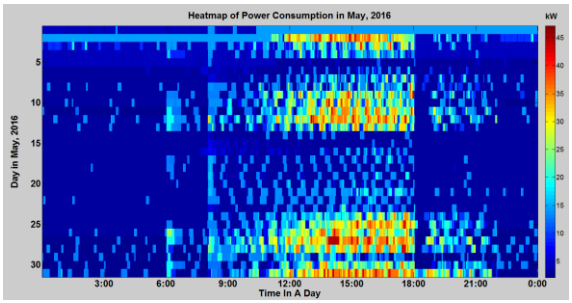

Fig. 5. Example of building operation power consumption heat map. X-axis represents time of the day; Y-axis represents dates in May 2016.

VII. Conclusion

This paper presents an example of how IoT devices can be integrated with an open source BEM systems to enable smart building operation. Specifically, BEMOSS, an open source BEM platform, is able to run on a SBC such as Odroid to control and monitor IoT devices in buildings. This provides a low cost and powerful solution for managing building operation, thus potentially enabling energy savings in commercial buildings for the first time. Test and evaluation of deploying BEMOSS on several SBCs show the Odroid XU4 is the most suitable embedded system for this application. Experience with IoT device integration and system deployment is summarized in this paper, which can serve as reference and guidance for future research and similar real-world implementations.

References

[1] Gartner, (2015, November) “Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015”. [Online]. Available: www.gartner.com/newsroom/id/3165317.

[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. "Internet of things for smart cities." IEEE Trans. on Internet of Things, vol. 1, no. 1, pp. 22-32, Feb 2014.

[3] W. He, G. Yan, and L. Xu. "Developing vehicular data cloud services in the IoT environment." IEEE Trans. on Industrial Informatics, vol. 10, no. 2, pp. 1587-1595, May 2014.

[4] F. Tao, Y. Zuo, L. Xu, and L. Zhang. "IoT-based intelligent perception and access of manufacturing resource toward cloud manufacturing." IEEE Trans. on Industrial Informatics, vol. 10, no. 2, pp. 1547-1557, May 2014.

[5] S. Babar, P. Mahalle, A. Stango, N. Prasad, and R. Prasad. "Proposed security model and threat taxonomy for the Internet of Things (IoT)." In International Conference on Network Security and Applications, pp. 420-429. Springer Berlin Heidelberg, 2010.

[6] Y. Li. "Design of a key establishment protocol for smart home energy management system." 2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), Madrid, 2013, pp. 88-93.

[7] S. Kelly, N. Suryadevara, and S. Mukhopadhyay. "Towards the implementation of IoT for environmental condition monitoring in homes." IEEE Sensors Journal, vol. 13, no.10, pp. 3846-3853, May 2013.

[8] J. Serra, D. Pubill, A. Antonopoulos, and C. Verikoukis. "Smart HVAC control in IoT: Energy consumption minimization with user comfort constraints." The Scientific World Journal, vol. 2014, pp. 1-11, June, 2014.

[9] Cho, Wei-Ting, Ying-Xun Lai, Chin-Feng Lai, and Yueh-Min Huang. "Appliance-aware activity recognition mechanism for IoT energy management system." The Computer Journal, May 2013.

[10] W. Khamphanchai, A. Saha, K. Rathinavel, M. Kuzlu, M. Pipattanasomporn, S. Rahman, B. Akyol, and J. Haack. "Conceptual architecture of building energy management open source software (BEMOSS)." In IEEE PES Innovative Smart Grid Technologies, Europe, Istanbul, Turkey, 2014, pp. 1-6.

[11] Source code for BEMOSS. [Online]. Available: https://github.com/bemoss/bemoss_os

[12] Gunther, Neil J. "Linux Load Average." Analyzing Computer System Performance with Perl PDQ. Springer Berlin Heidelberg, 2011. 215-238.

[13] Source code for Particle API. [Online]. Available: https://github.com/Alidron/spyrk

[14] Monitor your home temperature using your Raspberry Pi. [Online]. Available: http://projects.privateeyepi.com/home/temperature-gauge