SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM

TNC TRANSACTIONS ON NETWORKS AND COMMUNICATIONS

VOLUME 6, NO. 5
ISSN: 2054 -7420

Web Browser Based Data Visualization Scheme for XBee Wireless Sensor Network

[1]Xinzhou Wei, [2]Li Geng, [3]Xiaowen Zhang
[1,2]Department of Electrical and Telecommunications Engineering Technology,
New York City College of Technology, City University of New York, 300 Jay St, Brooklyn, NY 11201, U.S.A.
[3]Department of Computer Science, College of Staten Island, City University of New York
2800 Victory Blvd., Staten Island, NY 10314, U.S.A.
xwei@citytech.cuny.edu; lgeng@citytech.cuny.edu; xiaowen.zhang@csi.cuny.edu

ABSTRACT

Wireless sensor network (WSN) plays an important role in the infrastructure of Internet of Things (IoT). Data visualization is an essential component in WSN to facilitate data scientists to interpret information clearly and efficiently. In this paper, we conduct a study of XBee based WSN which integrates the DASH data visualization scheme for building a web-browser based application without using HTML or JavaScript. The data collected from wireless sensors in a WSN were displayed in a web browser with interactive functions. The proposed visualization scheme is real-time, cross platform, and hardware independent. Thus, it could be easily employed on any operating system. Experimental results demonstrated that our WSN data visualization scheme using XBee Python package and Plotly's DASH is feasible for IoT applications like smart buildings, environment monitoring, as well as other WSN applications.

Key words: Data visualization, Internet of Things, Smart building, Wireless sensor network, XBee, ZigBee.

1 Introduction

Wireless sensor network (WSN) has been widely used in the infrastructure of Internet of Things (IoT) [1-3]. It has been applied to the fields related to smart building management [4], environment monitoring [5], energy monitoring [6], health monitoring [7], and precision agriculture [8].

Data visualization is an appealing way to interpret the data in an illustrative and graphical form. It typically converts texts and numbers to aesthetically pleasing visual elements, thus makes them easy to be recognized by human beings [9]. This is the most important reason why data visualization is so compelling to data scientists, data engineers, and researchers. Compared to conventional text reading and number processing by human brain, data visualization can take the same information and make patterns more understandable and readily perceptible [10].

With increased demand in the application of WSN, data visualization has become a key component of sensor networks. As dozens or hundreds of sensors generate a large amount of data in WSN, a powerful visualization tool will help data scientists or decision makers to spot a special event or recognize some unique patterns quickly and efficiently. In the past decades, researchers have developed a lot of visualization tools for WSN, such as TinyViz [11], SpyGlass [12], MoteView [13], MeshNetics [14],

DOI: 10.14738/tnc.65.5261
Publication Date: 18th October 2018
URL: http://dx.doi.org/10.14738/tnc.65.5261

MonSense [15], NetTopo [16], WiseObserver [17], Octopus [18], and Surge Network Viewer [19], *etc.* Although these visualization tools are wonderful, most of them are stand-alone computer applications and have high demand for computer hardware and resources. Some of them were developed on a specific operating system or hardware platform. While others cannot display the data in real- time and are only suitable for static data. Furthermore, some of those visualization tools do not provide interactive functionalities to facilitate user-friendly interface [20-23]. Thus, there is a strong demand for a cross platform data visualization tool which shall display the real time data in WSN interactively. In addition, another objective of this study is to develop a powerful data visualization tool for WSN without the need of advanced programming techniques from the users. As a result, the DASH data visualization framework from Plotly is the best solution for the above criteria.

1.1 Current Data Visualization Tools in Python

The evolution of computers and digital technology makes data visualization possible to process large amounts of data in real-time with interactive rendering components [10]. One big challenge, particularly for data scientists or data engineers, is to represent the innovative ideas via visual means without concerning too much about programming skills and the platforms. Python, a simple open source and cross platform script language, fulfills the requirements for non-programming professionals. The users of Python in scientific fields increased exponentially in the past five years. There are a couple of cross platform visualization tools developed in Python in recent years, such as Pandas, Matplotlib, Seaborn, Bokeh, Pygal, PyQtGraph, Plotly, and VisPy [24]. These visualization tools have demonstrated their technological abilities in different technology fields.

1.2 Python DASH Visualization Scheme

The visualization tools mentioned above are excellent for data scientists or data engineers, but sometimes there is a strong need to create interactive visualizations and more dynamically explore data like surfing on the Internet via web browser [25]. Interactive data visualization tools allow users to explore and analyze data with greater freedom and flexibility. DASH is a cross platform, web browser based, and open source framework created by the Plotly team that leverages Flask, Plotly.js and React.js to build custom data visualization applications in Python [26]. DASH provides most attractive interactivity functions without using HTML or JavaScript. Users can manipulate data in a web browser based environment and can seamlessly take advantage of their experience on web surfing. It greatly speeds up the development cycle, simplifies the development difficulty, and shortens the learning curve for data scientists and data engineers [26].

1.3 Characteristics of Plotly DASH

DASH was recently released in June 2017 by the Plotly team as an open source library. Built on top of Plotly.js, React.js, and Flask, DASH is ideal for the users to develop web-based visualization application interactively [26]. Developers can use Python solely for building interactive web browser based applications. No HTML or JavaScript programming skill is required. Other characteristics of DASH include that developers can adopt all the plotting capabilities which are user-friendly via Plotly's Python framework, access other Python libraries such as Matplotlib or Pandas, and use all of its powers in their visualization tool impeccably. This open source model was adopted widely by data science and industry in recent years. Plotly makes this open source package available publicly on GitHub for those

individuals like data scientists, researchers, and college students [27]. The technical supports, such as training and large scale deployments, are also available from Plotly based on developer's requests [26].

The remainder of this paper is organized as follows: the configuration of XBee modules in WSN, WSN topology, the data collection method for the XBee based WSN, and the structure of Python DASH visualization scheme are presented in Section 2. Section 3 reports the experimental results of wireless temperature monitoring system for smart buildings. The conclusion and discussion are provided in Section 4.

2 Methods

In this study, we propose an XBee based WSN, in which Python DASH data visualization framework has been integrated to build a cross platform, web-browser based interactive WSN environmental monitoring system without using HTML or JavaScript. For the configuration of XBee modules in WSN, one XBee module works as the coordinator that acts like the root of a tree. The coordinator collects all information forwarded by routers in WSN and sends them to a computer via USB connection, whereas a router relays and forwards the information collected by different wireless end nodes in the WSN. Its configuration will be different from that of the coordinator. In addition, all XBee modules in the same WSN should contain the same PAN ID number. Detailed procedure of XBee coordinator and router configuration is described in Faludi's work [28].

Fig. 1 and Fig. 2 show the configurations of an XBee coordinator and a router, respectively, from the XCTU (a free application created by Digi International Inc. for the XBee module programming, configuration, troubleshooting, and network management.) Details of the XCTU application can be found at http://www.digi.com/products/wireless-wired-embedded-solutions/ZigBee-rf-modules/xctu. The parameters of WSN coordinator and router in XCTU configuration interface are shown in Fig. 1 and Fig. 2, respectively. The PAN ID is configured as 1 for both XBee coordinator and routers for the simplicity of demonstration purpose. XCTU is a pure graphic user interface and each parameter of XBee module could be set individually. User can set up a parameter by clicking the textbox on the right side of the parameter label where XCTU will pop up a menu and allow the user to select certain values in the menu. XCTU also uses different colors to indicate the type of the value.

2.1 The XBee Router Configuration

To configure the XBee module as a router, firstly, we need to mount the XBee module on an adapter called XBee explorer board. Developers can obtain the XBee explorer board from Sparkfun Electronic Inc. or Adafruit Inc. Secondly, we attach a mini USB cable from this adapter and connect it to a PC. There is an onboard FT231X USB-to-Serial converter that translates data between XBee and the PC. Finally, we start the XCTU that will automatically detect the XBee module connected via USB port. User could have an option to update the firmware of the XBee module after a successful detection. All routers must have the same PAN ID number with coordinator in the same WSN. Otherwise they will not communicate with each other. In our XBee router, the value of pin DIO0 is configured as ADC input port that has a value 2 as shown on the first textbox of right screen snapshot in Fig. 1.

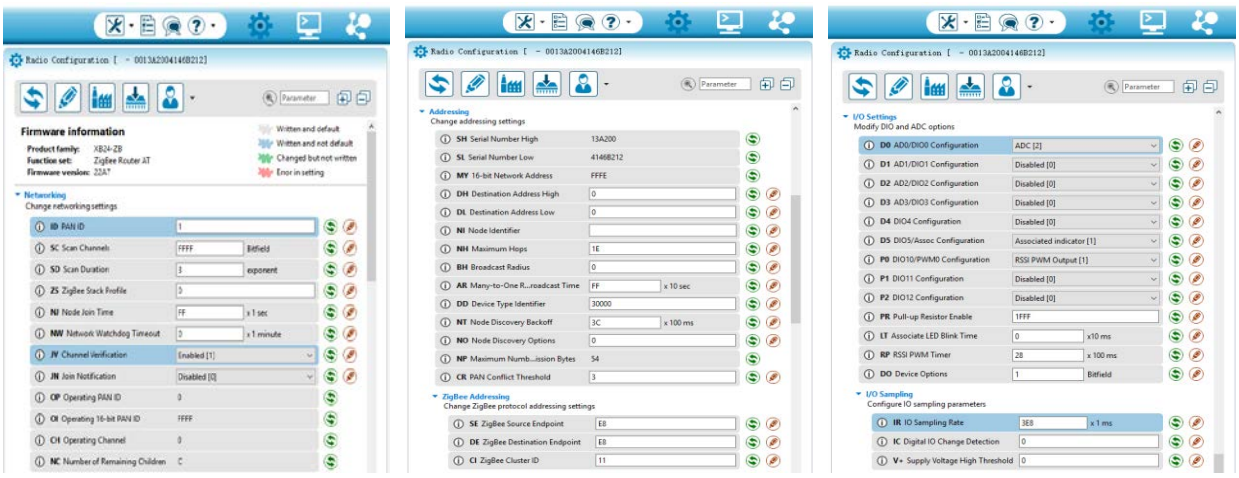

Fig. 1. Configuration of XBee Router: Network addresses setting (left and middle) and the I/O ports setting (right).

2.2 The XBee Coordinator Configuration

*The XBee coordinator configuration interface in Fig. 2 shows that the XBee module has been set to API mode and the PAN ID of the coordinator must have the same number with the XBee routers in the WSN.

Fig. 2. Configuration of XBee Coordinator: Network addresses setting (left and middle) and the I/O ports setting (Right)

There are two parts in the MAC addresses for both coordinator and routers: 1) The High part is always "13A200" which was assigned by Digi International Inc; 2) The Low address is a hexadecimal number based on that XBee module. The network addressing configuration values and I/O ports configuration values are shown in Fig. 2. In our WSN system, the pin of DIO0 for the XBee coordinator is configured as a digital input port, which has a value of 3 as shown on the first textbox of right screen snapshot in Fig. 2.

2.3 Wireless Sensor Network Topology

After we successfully configure both XBee coordinator and routers, we will test the connections and display the network topology of our XBee based WSN. The details of hardware are described in our earlier paper [29]. XCTU's Network function allows user to discover and visualize the topology and interconnections of a WSN. To display the topology of our WSN, we switch the setting to "Network working mode" and click the "Scan" radio button to start the network discovery process. XCTU will scan

the entire XBee based WSN and display the logical connections and link quality as shown in Fig. 3. For the demonstration purposes, in this paper, we only use a couple of XBee modules in our WSN system

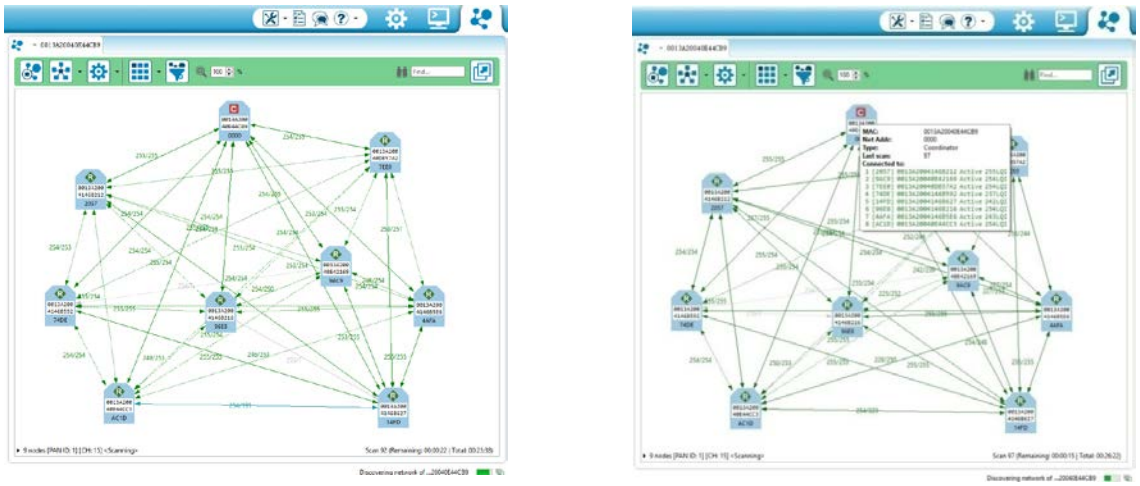

Fig. 3. Wireless sensor network topology and connection table in the XCTU Network Interface: WSN topology (left) and topology and connection table (right).

The coordinator is represented by letter 'C' in red color and the router is represented by letter 'R' in green color in the XBee icons. The wireless sensor network shown in Fig. 3 (left) is a real network and the links and their signal qualities are detected by XCTU. To find out the detailed connection information for a specific XBee module, we can click any XBee icon in XCTU network window, a connection table would pop up and display all of the connections with that XBee module as shown in Fig. 3 (right).

All these XBee modules must be XBee S2, XBee S2 Pro version or newer models that make a mesh wireless sensor network. It is noted that XBee S1 version only supports point to point communication and thus cannot be used in a mesh WSN.

After we finish setting up our WSN system, we will collect data and build a data visualization system using Plotly DASH to display data in a web browser.

2.4 Data Collection of XBee Based Wireless Sensor Network

In order to collect data from serial port in our WSN in Python, we need to include XBee, ZigBee, and serial packages. The tool to install these packages in Python IDE is called pip. In this study, we use PyCharm Community 2017, an open source Python IDE to manage our project. As shown in the code snippet of Fig. 4, a Python function called get_serialData() will read XBee data frame collected from the sensors via serial port. First, we called the xbee.wait_read_frame() function in Python XBee package distributed by Digi International Inc. and saved it in a variable "responseData."

```
def get_serialData():
    responseData = xbee.wait_read_frame()
    print("responseData['source_addr_long']: ", responseData['source_addr_long'])
    temperature = get_temperature(responseData['samples'], format="F")
    temp = float(temperature)
    return temp
```

Figure 4. Code snippet of XBee module data collection from serial port.

Then, we applied a user defined function called get_temperature() to extract the temperature and convert it in Fahrenheit. The code snippet of get_temperature() function is listed in Fig. 5.

```
#get the current temp from a list of voltage readings
def get_temperature(data, format="C"):
    readings = [ ]
    for item in data:
        readings.append(item.get('adc-0'))
    #start by averaging the data
    volt_average = sum(readings)/float(len(readings))
    #now calculate the proper temperature
    temperature = (volt_average / 1023.0 * 1.2 * 3.0 * 100) - 273.15
    if format=="F":
        #convert to farenheit
        temperature = (temperature * 1.8) + 32
    return temperature
```

Fig. 5. Code snippet of XBee module temperature conversion from XBee data frame.

In the code snippet of the function get_temperature(), the temperature was calculated by the following temperature sensor's formula:

temperature = (volt_average/1023*1.2*3*100) - 273.15

where volt_average is the analog input of the sensor. Since there are a couple of temperature sensors in our WSN system, we can extract both XBee's MAC addresses and the corresponding temperature values in the XBee frame to distinguish different sensors before performing data visualization.

2.5 Structure of the Python DASH Visualization Scheme

In this paper, we employed the DASH visualization scheme, which is an open source library using Python framework to build web browser based applications. There is no HTML or JavaScript needed in this structure. Written on top of Flask, Plotly.js, and React.js, DASH is ideal for building data visualization applications with better user interfaces in pure Python [30]. With DASH, we can create cross platform, web-based interactive applications in pure Python. It is particularly suitable for data scientists or researchers who work in data visualization field in Python. Specifically, Plotly keeps a set of visual components in dash_core_components and dash_html_components library. We imported these packages for developing DASH visualization applications in this study.

2.5.1 DASH layout

There are two parts in the Python Dash Visualization application. The first part is the layout of the application and it decides what the application looks like. The second part decides the interactivity of the DASH application. Furthermore, the DASH layout could be divided in three blocks: Header, Dropdown menu, and Graph. It is composed of a tree of components like html.Div and dcc.Graph. The dash_html_components library has a component for each HTML tag.

In this study, we adopted some codes publicly available to researcher & developer [27][30]. As shown in the code snippet of Fig. 6, the header of the webpage could be created by html.Div and html.H1 with suitable style. The html.H1(children = 'Wireless Network Building Temperature Monitoring System') component generates a HTML element (<h1>Wireless Network Building Temperature Monitoring System </h1>) in the DASH application. The contents in the Dropdown menu will be decided by dcc.Dropdown

block with corresponding id value and keys in the dictionary. The display interval and style could also be decided in this step.

```
app.layout = html.Div([

    html.Div([html.H1('Wireless Network Building Temperature Monitoring System',
                      style={'float': 'center', }), ]),
    dcc.Dropdown(id='room-temp-data', options=[{'label': s, 'value': s}
                 for s in data_dict.keys()],
            value=['Room1 Temperature','Room2 Temperature','Room3 Temperature'],
            multi=True),
    html.Div(children=html.Div(id='graphs'), className='row'),
              dcc.Interval( id='graph-update', interval=100), ],  className="container",

    style={'width':'90%','margin-left':15,'margin-right':15,'max-width':50000})
```

Fig. 6. Code snippet of DASH layout.

2.5.2 DASH core components

The dash_core_components includes a set of higher-level components like dropdown, graph, markdown block, etc. Graph renders interactive data using the open source Plotly.js, a JavaScript graphing library. Plotly.js supports over 35 chart types and renders charts so far in both vector-quality SVG and high-performance WebGL[26]. Graph component is the same figure argument that is used by Plotly.py, a Plotly's open source Python graphing library [26].

2.5.3 DASH interaction method

After the DASH layout is created, we map out the interaction among various DASH components. DASH provided app.callback() decorator to fulfill this requirement. We employed DASH "callback" to bind interactive components such as dropdowns, graphs, sliders, and text inputs in its application. As shown in the code snippet of Fig. 7, the parameters we pass into the app.callback decorator include output components and properties we want to update plus a list of all the input components and properties that can be used to trigger the function.

```
@app.callback(
        dash.dependencies.Output('graphs','children'),
        [dash.dependencies.Input('room-temp-data', 'value')],
        events=[dash.dependencies.Event('graph-update', 'interval')])
```

Fig. 7. Code snippet of DASH callback.

With DASH interactivity, we can dynamically update any of those properties through the callback function. That is to say, we could not only update the children's values of a component and display new text or figure of a dcc.Graph component, but also update the style of the component or even the option values of a dcc.Dropdown component. After laying out all of the above components, we define the figure using a dictionary that contains the figure as well as the data and layout options. Details of these configurations could be found at www.pythonprogramming.net [27].

2.5.4 Launch of the DASH Application

As shown in Fig. 8, we provided the local host's IP address and the port number for the browser to make sure it can find and display the data in the proper location when we launch the DASH application. We launched a web browser and entered address 127.0.0.1:8000 in the address box. The real-time data collected from XBee based WSN will be displayed in web browser shortly and refreshed periodically.

```
if __name__ == '__main__':
    #ADDRESS="127.0.0.1"   PORT = 8000
    app.run_server(port=PORT, host=ADDRESS)
```

Fig. 8. Code snippet of launching project.

3 Experimental Results

The XBee based DASH visualization scheme for WSN could be employed on any web browser like Internet Explorer, Firefox, or Google Chrome. We tested our visualization system displaying the temperature values from different rooms in a smart building with screen snapshots as shown in Fig. 9. For the arrangement of the display, when there's only one graph, DASH will take entire row of the browser. If we select to display two values in the dropdown menu from top, then each value will take half size of the browser horizontally. When we select more values to display, DASH will arrange the graphics in a matrix format.

As shown in Fig. 10, when a user performs a mouse over on a specific graph, DASH will show a group of interactive tools, such as dropdown menu, text box, zoom in, zoom out, auto size, download figure, expand the size of figure, etc. in the browser to display the value in a pop-up text box and show the temperature value. We could zoom in and zoom out over a selected area and check more detailed information. We could also deselect a specific graph from dropdown menu and remove it from visualizing.

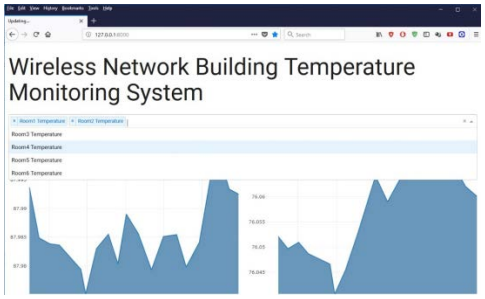

Fig. 9. Visualization system displaying the temperature values from different rooms in a smart building.

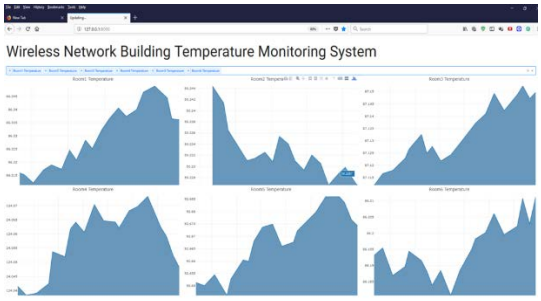

Fig. 10. Visualization system displaying room temperature values with interactivity function.

If we prefer to set up the computer monitor vertically from pivot, we can display all values vertically as shown in Fig. 11.

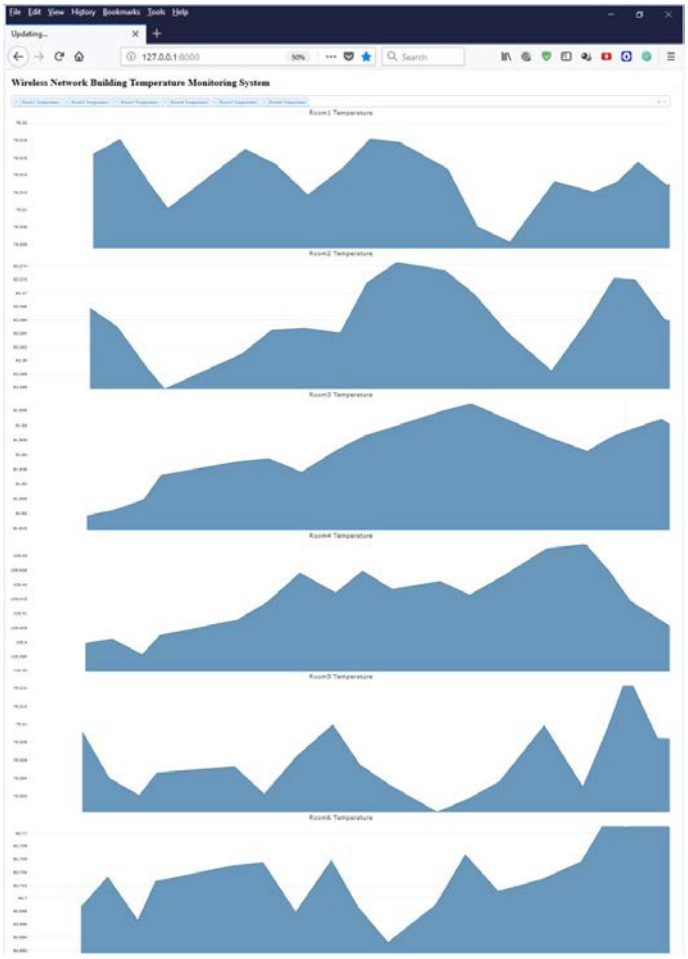

Fig. 11. Visualization system displaying values vertically in the browser with a snapshot of full screen.

4 Conclusions and Discussion

In this paper, we present a web browser based data visualization scheme for WSN. Experiment results demonstrated that the data collected from wireless sensors in a WSN were displayed in a web browser with interactive functions. We can select the temperature data from different rooms in a drop-down menu and visualize them in real- time. We have the option to add or remove some graphs to display in the drop-down menu. Finally, we can choose the layout on the page depending on how many charts we want to display.

Our work using this DASH visualization scheme can be extended by further improvement of uploading data of WSN on cloud and then scraping down to any local machine for visualization in a web browser. Future improvements include adoption of X4 Portconnect and wireless nodes from Digi International Inc. to simplify the hardware and improve the reliability of the system.

ACKNOWLEDGEMENT

This work was partly supported by CUNY GRTI grant round 20 and PSC-CUNY grant #61163-00 49, jointly funded by the City University of New York.

REFERENCES

[1] Minoli D, Sohraby K, Occhiogrosso, B. IoT considerations, requirements, and architectures for smart buildings—Energy optimization and next-generation building management systems. IEEE Internet of Things, 2017, 4(1): 269-283.

[2] Zanella A, Bui N, Castellani A, Vangelista L, Zorzi M. Internet of Things for smart cities. IEEE Internet of Things, 2014, pp. 22-32.

[3] Mitton N, Papavassiliou S, Puliafito A, Trived K S. Combining cloud and sensors in a smart city environment. EURASIP Journal on Wireless Communications and Networking, 2012, p. 247.

[4] Ghayvat H, Mukhopadhyay S, Gui X, Suryadevara N. WSN- and IOT-based smart homes and their extension to smart buildings. Sensors, 15(5): 10350–10379.

[5] Jang W S, Healy W M, Skibniewski M J. Wireless sensor networks as a part of a web-based building environmental monitoring system. Automation in Construction, 2008, 17(6):729-736.

[6] Liu X, Chen H, Wang M, Chen S. An XBee-Pro Based Energy Monitoring System. Brisbane, Australasian Telecommunication Networks and Applications Conference (ATNAC), 2012, pp. 1-6.

[7] Othman S B, Trad A, Youssef H. Security architecture for at-home medical care using wireless sensor network.. International Wireless Communications and Mobile Computing Conference (IWCMC), 2014, pp. 304-309.

[8] Davcev D, Mitreski K, Trajkovic S, Nikolovski V, and Koteli N. IoT agriculture system based on LoRaWAN. 14th IEEE International Workshop on Factory Communication Systems (WFCS). 2018.

[9] Shamas N. Why data visualization is important. TechChange, May 19, 2015. https://www.techchange.org/2015/05/19/data-visualization-analysis-international-development/.

[10] Link A. Why create visualizations of your data? Data Visualization, 2017. https://dash.umn.edu/data-visualization/.

[11] Levis P, Lee N, Welsh M, Culler D. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In Proc. the 1st International Conference on Embedded Networked Sensor Systems, 2003, pp. 126-137.

[12] Buschmann C, Pfisterer D, Fischer S, Fekete S P, Kroller A. SpyGlass: a wireless sensor network visualizer. ACM SIGBED Review, 2005, 2(1): 1-6.

[13] Tuton M. MOTE VIEW: A sensor network monigoring and management tool. Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II), 2005, pp. 11-18.

[14] Leonov A. MeshNetics demonstrated integration of wireless sensor data with SCADA system at ZigBee open house. CISION - PRWeb, June 2006. https://www.prweb.com/releases/2006/06/prweb403245.htm.

[15] Pinto J, Sousa A, Goncalves G M, Lebres P, Sousa J. MonSense-application for deployment, monitoring and control of wireless sensor networks. ACM Real Wireless Sensor Network Conference, 2006.

[16] Shu L, Wu C, Zhang Y, Chen J, Wang L, Hauswirth M. NetTopo: Beyond Simulator and Visualizer for Wireless Sensor Networks. IEEE Second International Conference on Future Generation Communication and Networking, 2008, pp. 17-20.

[17] Castillo J A, Ortiz A M, Lopez V, Olivares T, Orozco-Barbosa L. WiseObserver: a real experience with wireless sensor networks. In Proc. the 3nd ACM workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, 2008, pp. 23-26.

[18] Jurdak R, Ruzzelli A G, Barbirato A, Boivineau S. Octopus: monitoring, visualization, and control of sensor networks. Wireless Communications & Mobile Computing, 2011, 11: 1073-1091.

[19] Surge Network Viewer. By Crossbow Technology, Inc. http://www.hoskin.qc.ca/uploadpdf/Instrumentation/divers/CrossBow/divers_Surge%20Network%20Viewer_4271286a0135f.pdf.

[20] Parbat B, Dwivedi A K, Vyas O.P. Data visualization tools for WSNs: A glimpse. International Journal of Computer Applications, 2010, 2(1): 14-20.

[21] dAuriol B J, Lee S, Lee Y. K. Visualizations of Wireless Sensor Network Data. In Handbook of Research on Developments and Trends in Wireless Sensor Networks: From Principle to Practice. Hershey : IGI Global, 2010, Chapter 16, pp. 353-370.

[22] ElHakim R, ElHelw M. Interactive 3D visualization for wireless sensor networks. 6-8, June 2010, The Visual Computer, 2010, Vol. 26, Issue 6-8, pp. 1071-1077.

[23] Ravichandranb S, Chandrasekarb R K, Uluagac A S, Beyah R. A simple visualization and programming framework for wireless sensor networks: PROVIZ. Ad Hoc Networks, 2016, 53: 1-16.

[24] Moffitt C. Overview of Python visualization tools. Practical Business Python, 2015. http://pbpython.com/visualization-tools-1.html.

[25] Moffit C, Choosing a Python visualization tool. Practical Business Python, 2018. http://pbpython.com/python-vis-flowchart.html.

[26] Moffitt, C. Creating interactive visualizations with Plotly's Dash framework. Practical Business Python, 2017. http://pbpython.com/plotly-dash-intro.html.

[27] Kinsley H. Introduction to data visualization applications with Dash and Python. Pythonprogramming.net, 2018. https://pythonprogramming.net/data-visualization-application-dash-python-tutorial-introduction/.

[28] Faludi R. Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing. O'Reilly Media, 2010.

[29] Wei X, Geng L, Zhang X. An open source data visualization system for wireless sensor network. Journal of Computer Science and Information Technology, 2017,5(2): 10-17.

[30] Parmer C. Introduction to DASH. Build beautiful web-based interfaces in Python. 2016. https://dash.plot.ly/introduction.