ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPL007

EPICS CONTROLLED WIRELESS SENSORS

M. Rolland[1], K. J. Gofron†, Brookhaven National Laboratory, Upton, USA
[1]also at Stony Brook University, Stony Brook, USA

Abstract

For sensor application, wireless technologies are more portable and convenient than their wired counterparts. This is especially true in scientific user facilities, where many environmental factors must be recorded at many locations simultaneously during data collection. Using wireless technologies, scientists can often reduce cost while maximizing the number of sensors without compromising sensor quality. To this end, we have developed EPICS controllers for both Bluetooth Low Energy (BLE) sensors and Zigbee sensors. For BLE, we chose the Nordic Thingy:52 for its low cost, high battery life, and impressive range of sensors. The controllers we developed combine EPICS base functions, the Bluetooth generic attribute data structure library, and multithreading techniques to enable real-time broadcast of the Thingy's 20+ sensors' live values. Because BLE is limited in range, we also developed a controller for an XBee sensor which, through the Zigbee mesh protocol, can expand its range through each node added into the network. With these controllers, NSLS-II scientists have access to a whole new class of sensors which are both easier to deploy and cheaper than their wired predecessors.

INTRODUCTION

At the beamlines of Brookhaven National Laboratory's National Synchrotron Light Source II, scientists have countless environmental parameters that they want to monitor and control. Using wired sensors is a popular solution, but requires fiddling with wiring, ports, converters, and typically several protocols. Wireless sensors would remove all the monotonous tinkering while still providing the accurate sensing that scientists require. Thus, we set out to create Experimental Physics and Industrial Control System (EPICS) [1] input-output controllers (IOCs) that scientists at the Synchrotron may use to interface with wireless sensors that are supremely simple to deploy. Along with the IOCs, we developed intuitive screens for the Control Systems Studio [2] client to help users visualize data and interact with the IOC software. Note that our goal is not to maximize wireless range, but to create a wireless sensing system that is durable, portable, cost-effective, long-lasting and simple to deploy. The primary purpose of such a system is to initially replace wired sensors in an individual beamline hutch, and thus does not require extensive long-range communication.

ZIGBEE: DIGI XBEE L/T/H

The first wireless sensor we considered was the XBee L/T/H sensor [3], a sensor produced by Digi which utilizes the Zigbee protocol and senses light, temperature, and humidity. Figure 1 shows the sensor as well as the peripherals we used to interface with the sensor. We were interested in a Zigbee sensor due to the protocol's low power, high battery life, and mesh networking support.

Figure 1: An XBee L/T/H sensor, Digi Wi-Fi Gateway, and XBee XStick.

Mesh networking is a local network topology in which every node dynamically and non-hierarchically communicates with other nodes to efficiently route data. The communication is forwarded through nodes to reach far away nodes. In effect, mesh networking allows for an extensible network with virtually unlimited range given enough nodes.

The first step in creating the controller for the sensor was to bridge the gap between the host device running the IOC and the sensor. With the Digi XBee sensors, we found ourselves locked into Digi's ecosystem; they used a proprietary Zigbee protocol that forced us to use their gateway hardware to communicate with the sensors. For our development we used an XBee Zigbee Wi-Fi Gateway [4], which we could connect to through Ethernet and configure our sensors. By default, the Gateway simply publishes sensor readings to a paid cloud service provided by Digi; this was unsuitable for our needs, as we only wanted to store and broadcast the readings locally. We found that the Gateway runs Python scripts with a proprietary package created by Digi to allow reading of the sensors, so we developed a script which runs a TCP server on the Gateway that receives simple commands and responds with sensor readings. We found that the easiest way to develop an IOC to interface with this server would be by utilizing StreamDevice, an IOC structure which uses simple protocols to read and write streams of bytes to a socket through EPICS. With the server script set to run indefinitely on the Gateway and using a custom XBee protocol file for our input-output controller, we had successfully created an EPICS system to read, and broadcast the readings from our sensors.

† kgofron@bnl.gov

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPL007

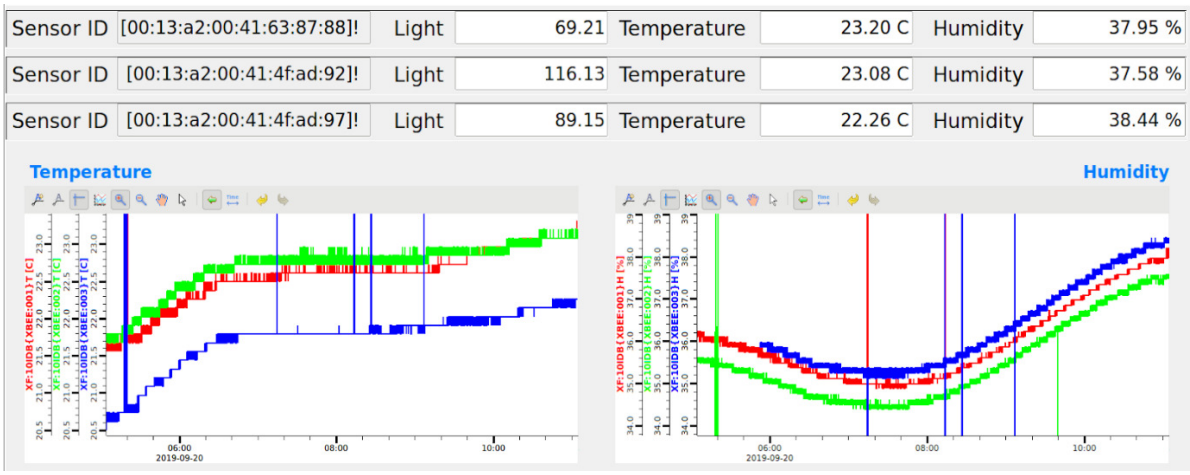

Figure 2: A Control Systems Studio screen displaying readings from three XBee sensors over six hours.

Although the Gateway was an effective solution, it was not the portable solution we were looking for as it required its own power and Ethernet; we instead looked to the XBee XStick [5], a USB dongle that supports Digi's proprietary communication. We found that the most effective way to interface with the dongle was through a proprietary Python package distributed by Digi [6], and we developed a similar script which creates a local TCP server to read the remote sensors which would interface with the same controller. Figure 2 depicts a CSS screen we developed along with the controller to visualize readings from all connected sensors. Although the Gateway and XStick solutions allowed us to read the sensors, there were several issues: interacting with an independent Python script was prone to failure, we were locked in to Digi's software and could not generalize the controller [7], and the sensors themselves were powered by three non-rechargeable AA batteries, not as compact as their alternatives and came at a hefty cost of $100 each. The advantage of a mesh protocol and extensive battery life was compelling, but we were looking for something cheaper, more portable, and more generalized.

BLUETOOTH: NORDIC THINGY:52

Bluetooth is by far one of the most popular wireless protocols and is used by countless sensors. Additionally, since the release of Bluetooth 4 the protocol has also supported low energy operation (BLE) which allows for extensive battery life without any compromise in fidelity. These factors made the BLE protocol a desirable choice, and we identified a suitable sensor: The Thingy:52 by Nordic [8]. Figure 3 shows the sensor (with its rubber cover removed) along with a Nordic development kit and dongle we used for development.

The Thingy:52 is a compact suite of environmental and motion sensors, which utilizes generic Bluetooth to broadcast data. Additionally, the Thingy:52 has a configurable RGB LED, button, speaker, and microphone. Compared to the XBee sensor, the Thingy:52 is smaller, less than half the price at $40, has far more sensors, and is powered by a lithium battery rechargeable with micro USB. To interface with the sensor, we used a $10 Nordic Bluetooth NRF52840 [9] dongle. The dongle comes pre-installed with bootloader firmware; however, we flashed Bluetooth Zephyr firmware onto the stick to make it compatible with generic Bluetooth libraries. In researching the Bluetooth protocol, we discovered that data is transferred through generic attributes (GATT characteristics) which are identified by universally unique IDs (UUIDs). These characteristics come in several types; however, the Thingy:52 characteristics were either read/write or notify. Read/write characteristics are self-explanatory, while notify characteristics must be 'subscribed' to by the client in order to receive data periodically. We developed our EPICS controller in C, using the gattlib library [10] developed by Lab A Part to manage the Bluetooth connection. Documentation from Nordic helped us identify the UUID and byte order for each sensor, and with gattlib we could create a connection and subscribe to each sensor individually.

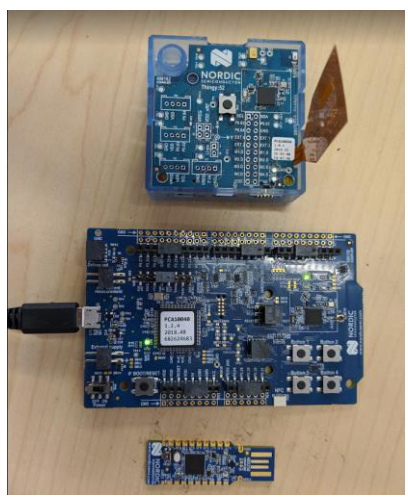

Figure 3: From top to bottom: A Thingy:52, NRF52 DK, and NRF52840 dongle.

The data sent by each characteristic does not contain any identifiers, such that a single thread handling notification from every sensor would not be able to distinguish which sensor sent any given packet of data. Thus, we took a multithreaded approach in developing the controller, dedicating a thread to each sensor so that there is no ambiguity in interpreting the received data. Figure 4 shows a CSS screen

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPL007

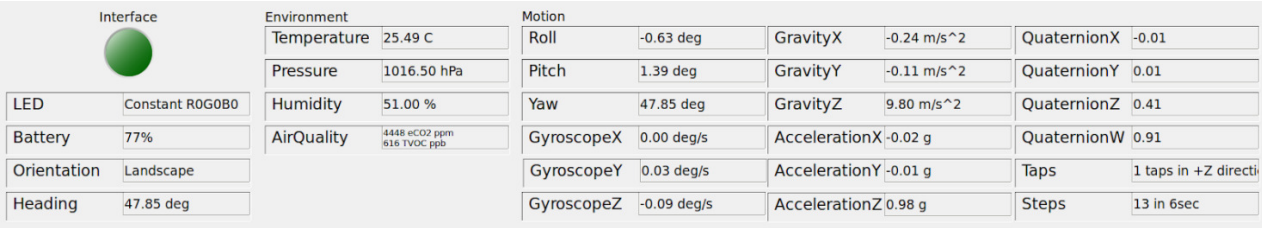

Figure 4: A Control Systems Studio screen displaying the readings of a single Thingy:52.

we developed to display sensor readings including battery, button, LED, temperature, pressure, humidity, air quality (equivalent CO2 and total volatile organic compounds), roll, pitch, yaw, orientation, acceleration in three dimensions, gyroscope in three dimensions, quaternions, heading, taps, and pedometer. We successfully developed a controller [11] that could read all the sensors of the Thingy:52, however we were limited by the one-to-one nature of Bluetooth; that is, with one dongle we could only connect to one Thingy:52 at a time. This was a major drawback, as monitoring large areas requires the usage of several sensors and we could not afford to sacrifice a USB slot for each sensor. We needed a protocol that, like Zigbee, would allow us to use one dongle to connect to many sensors.

Bluetooth Mesh

Bluetooth Mesh is a protocol which allows many-to-many connections over Bluetooth, creating networks which expand their range through each node and in which nodes can send data to any other node. We knew from Nordic's documentation that the Thingy:52 supported Bluetooth Mesh, however it would require custom firmware. Flashing new firmware onto the Thingy:52 required the use of a Nordic's NRF52 DK [12] development kit. Nordic also supplies two types of firmware for the Thingy:52 to create a mesh network: node firmware and bridge firmware [13]. The bridge Thingy:52 acts as an entry point for the client to receive readings from the network and coordinates the nodes to forward their readings. The node Thingy:52s simply act as regular sensors that forward their readings to the bridge through the mesh network. The firmware supplied by Nordic supported only temperature and humidity readings, so we modified it to also support pressure and battery level readings [14]. Figure 5 depicts the CSS screen we developed to simultaneously show the status of all nodes in the network, which during development was four. Since all readings were transmitted by the bridge through a single Bluetooth characteristic, development of the EPICS controller was rather simple, and all readings could be handled by a single thread. Additionally, the single-threaded reading design allowed us to create auxiliary threads that could monitor and fix the connection to the bridge in real time. The controller we developed [15] is able to read from up to 10 node Thingy:52s from one Bluetooth dongle; however, the issue of battery life became apparent within days of running the controller. Mesh networking requires the radios of the Thingy:52s to be running near constantly, resulting in the batteries draining completely in mere days. This issue could be avoided by keeping the Thingy:52s plugged into a power source at all times, but that would defeat the purpose of using wireless sensors. We needed a protocol that would allow a single dongle to connect to multiple Thingy:52s and would also allow the nodes to remain in low-power mode as they transmitted data. With help from Nordic, we discovered a solution: multi-link start networking.

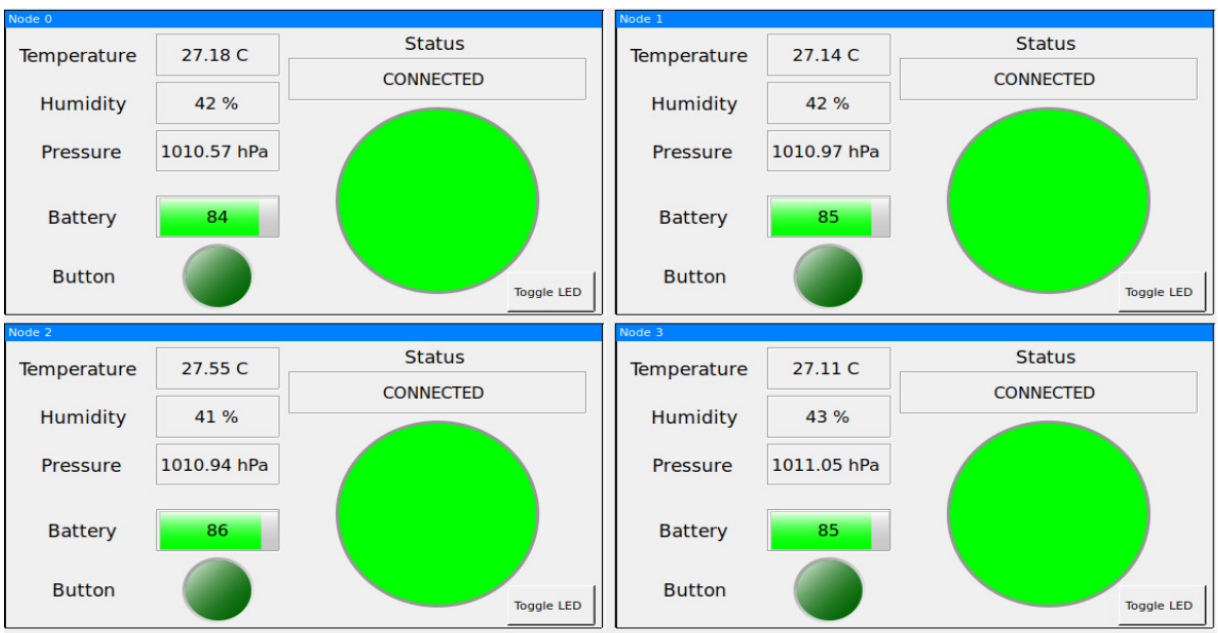

Figure 5: A Control Systems Studio screen displaying readings from four Thingy:52s in a mesh network.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPL007

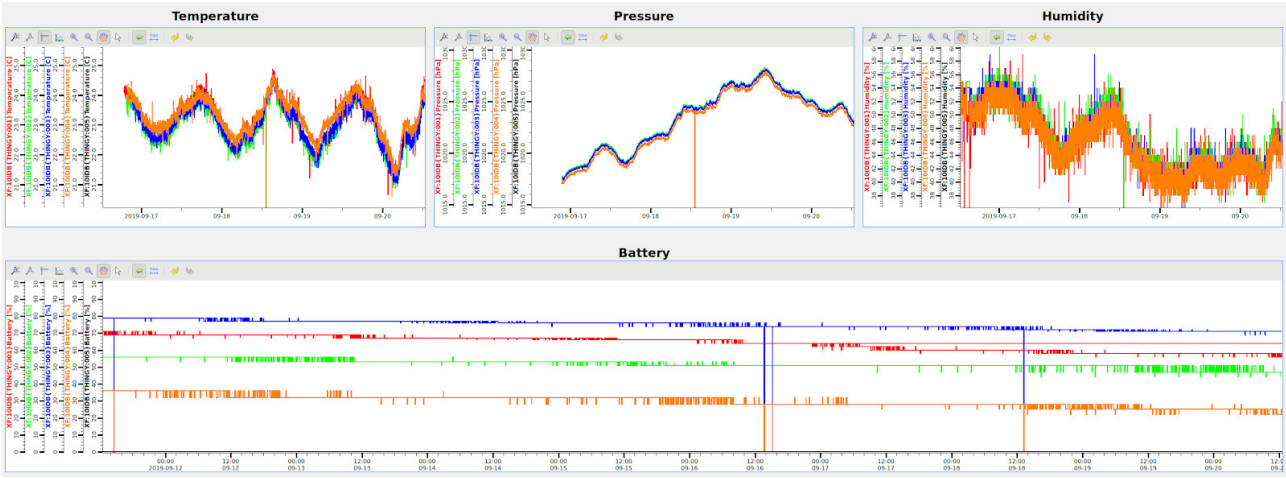

Figure 6: A Control Systems Studio screen displaying readings from four Thingy:52s in a multi-link network over several days.

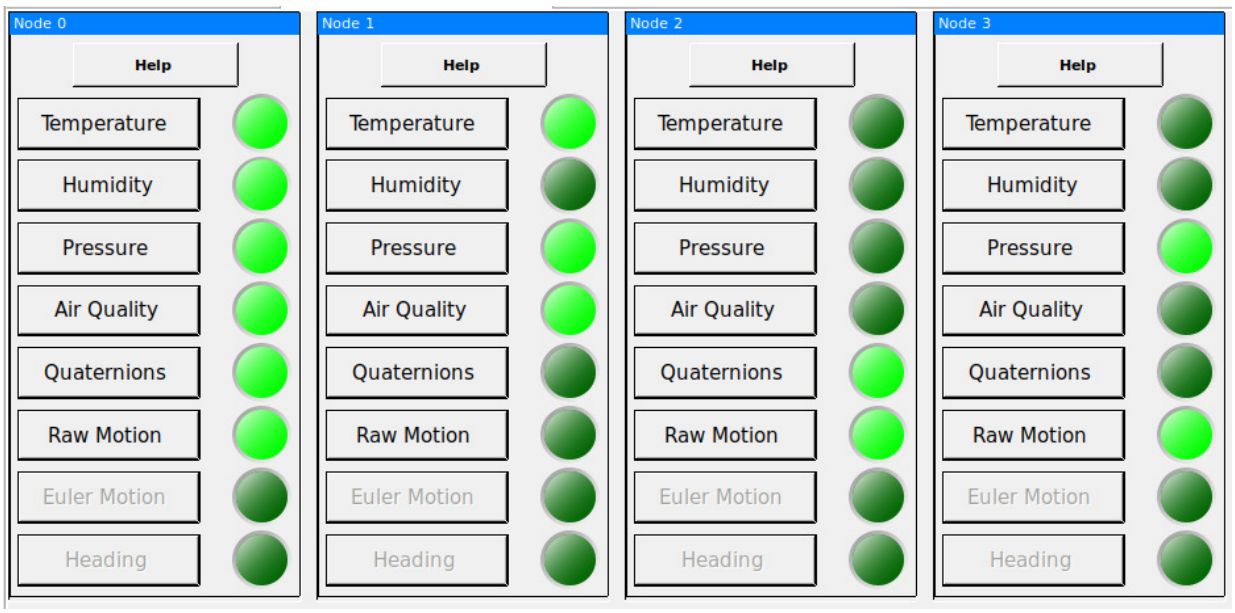

Figure 7: A Control Systems Studio screen displaying sensor toggles for Thingy:52 nodes in a multi-link network.

Bluetooth Multi-link

Firmware provided by Nordic [16] allows the NRF52 DK to become a central aggregator for a star network of sensors such as the Thingy:52. This network structure removes the overhead of mesh communication, and instead works with the Thingy:52's default firmware which implements low-power one-to-one communication. The firmware provided by Nordic allows the DK to connect to up to nineteen Thingy:52 nodes and a host to receive the sensor readings. In addition, the firmware supported a limited amount of commands for grouping nodes and toggling their LEDs from a provided Android app. Since the network utilizes the Thingy:52's default firmware; reading the sensors was achieved using standard Bluetooth communication. We implemented transmission of the Thingy Weather Station sensors, Thingy Motion sensors, battery level, and RSSI of the connection between the aggregator and the node. Figure 6 shows a CSS screen we developed to plot sensor readings from all nodes over time, in this case over 4 days for the environmental sensors and 9 days for the battery sensor for 4 nodes. Additionally, we implemented commands for writing the configuration characteristics (such as reading interval) for these sensors, reading connection parameters, and toggling individual sensors [17]. Figures 7, 8, and 9 show CSS screens we developed for interacting with these features. This customizability allowed us to easily optimize battery life by lowering scanning periods and disabling sensors which we didn't need. With all Weather Station sensors enabled (temperature, humidity, pressure, air quality) at their highest scanning period, we observed a loss of roughly 1% of battery every 24 hours on average. This was a vast improvement over the mesh network battery life; however, it did come with the loss of long range capabilities. In addition to being power-efficient, we were able to enable customization to this controller [18], which made it a very compelling solution.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPL007

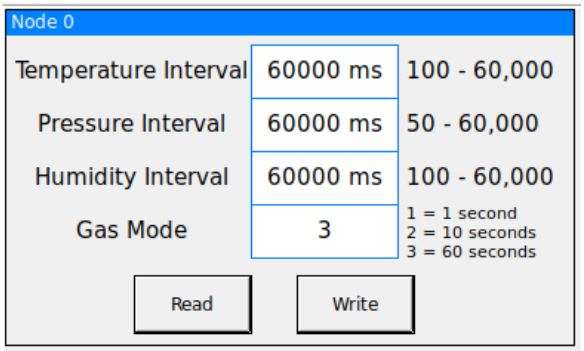

Figure 8: A Control Systems Studio screen displaying environment sensor config for a Thingy:52 node in a multi-link network.

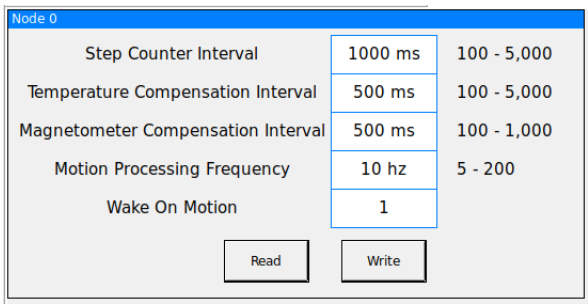

Figure 9: A Control Systems Studio screen displaying motion sensor config for a Thingy:52 node in a multi-link network.

CONCLUSION

Using Bluetooth multi-link, we have created an IOC which supports up to 19 wireless sensors for roughly 100 days on a fully charged battery. These sensors include temperature, humidity, pressure, and air quality as well as several motion sensors which may have applications in motor control and vibration sensing. These sensors are deployed by connecting a NRF52 DK to a computer for power, and then simply placing the Thingy:52s where needed. The EPICS controller is lightweight, requiring only EPICS base [1] and a Bluetooth C library [10], and can be deployed on compact hosts such as a Raspberry Pi equipped with a small Bluetooth USB dongle. This solution eliminates any need for wires and offers effective customizable monitoring of the environment, although the sensors must be recharged after roughly three months of continuous usage.

Bluetooth multi-link may not have the range of Zigbee or Bluetooth mesh, but we feel that the extended battery life justifies the trade-off. Our deployments will likely be limited to individual beamline hutches, which do not require incredible range and of course would benefit from maximizing battery life. The range of Bluetooth version 4 is rated at 200 feet (though it is also heavily dependent on the device and can be blocked by physical objects in the line-of-sight between the devices) which is sufficient for our use case; however, development of controllers for more capable mesh sensors may be the next step in developing wireless solutions on a facility-wide scale.

ACKNOWLEDGEMENT

This research used resources of the National Synchrotron Light Source II, a U.S. Department of Energy (DOE) Office of Science User Facility operated for the DOE Office of Science by Brookhaven National Laboratory under Contract No. DE-SC0012704.

REFERENCES

[1] EPICS Collaboration, https://epics.anl.gov/

[2] Control Systems Studio, http://controlsystemstudio.org/

[3] Digi XBee Sensors, https://www.digi.com/products/networking/rf-adapters-modems/xbee-sensors

[4] Digi XBee Gateway, https://www.digi.com/products/networking/gateways/xbee-gateway

[5] XStick USB Adapters, https://www.digi.com/products/networking/rf-adapters-modems/xstick

[6] Digi XBee Python Library, https://github.com/digidotcom/xbee-python

[7] XBeeIOC, https://github.com/epicsNSLS2-sensors/XBeeIOC

[8] Nordic Thingy:52, https://www.nordicsemi.com/Software-and-Tools/Prototyping-platforms/Nordic-Thingy-52

[9] nRF52840 Dongle, https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52840-Dongle

[10] gattlib, https://github.com/labapart/gattlib

[11] ThingyIOC, https://github.com/epicsNSLS2-sensors/ThingyIOC

[12] nRF52 DK, https://www.nordicsemi.com/?sc_itemid=%7BF2C2DBF4-4D5C-4EAD-9F3D-CFD0276B300B%7D

[13] Thingy Mesh Demo, https://github.com/NordicPlayground/Nordic-Thingy52-mesh-demo

[14] ThingyMesh, https://github.com/RollandMichael7/ThingyMesh

[15] ThingyMeshIOC, https://github.com/epicsNSLS2-sensors/ThingyMeshIOC

[16] Bluetooth 5 Multi-Link Demo, https://github.com/NordicPlayground/nrf52-ble-multi-link-multi-role

[17] Expanded Bluetooth 5 Multi-Link Demo, https://github.com/RollandMichael7/nrf52-ble-multi-link-multi-role

[18] ThingyAggregatorIOC, https://github.com/epicsNSLS2-sensors/ThingyAggregatorIOC