Expertise
and insight
for the future

Bijaya Rakhal

Environmental Monitoring with Nordic Thingy:52

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Information Technology

Bachelor's Thesis

13 November 2019

Author Title Number of Pages Date	Bijaya Rakhal Environmental Monitoring with Nordic Thingy:52 28 pages + 0 appendices 13 November 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Networking
Instructor	Marko Uusitalo, Senior Lecturer

This paper reviews the developments in NB-IoT and BLE technologies to develop a live environmental monitoring application. Furthermore, the guidelines for developing such systems as well as the impact of environmental monitoring on policymaking are discussed. Likewise, the potential application areas of NB-IoT and BLE environmental monitoring with Nordic Thingy:52 are detailed. Finally, the development of a remote home environment monitoring application capable of serving live readings from Nordic Thingy:52 is described. This paper also provides insights into ongoing and potential development areas of the application.

Keywords	IoT, BLE, Nordic Thingy:52, environmental monitoring, IEEE 801.15.4

Contents

List of Abbreviations

3GPP	Third Generation Partnership Project
AES	Advance Encryption Standard.
AFH	Adaptive Frequency-Hopping.
BLE	Bluetooth Low Energy.
CBC	Cipher Block Chaining.
CCM	Counter with CBC-MAC.
eSIM	embedded Subscriber Identity Module
HAL	Hardware Access Layer
ICT	Information and Communication Technology.
IFFT	If This Then That
IoT	Internet of Things.
KPI	Key Performance Indicators
LPWAN	Low Power Wide-Area Networks.
LTE-M	Long Term Evolution category M1
MAC	Message Authentication Code.
MCL	Maximum Coupling Loss
mMTC	massive Machine-Type Communication.

NB-IoT	Narrowband IoT.
OFDMA	Orthogonal Frequency-Division Multiple-Access.
PRBs	Physical Resource Blocks.
PSD	Power Spectral Density.
SCENTS	Sensing Collaboratively in Everyday Networks
SC-FDMA	Single-Carrier Frequency-Division Multiple-Access.
SDK	Standard Development Kit
UE	User Equipment
UMTS	Universal Mobile Telecommunication System.

1 Introduction

As Internet of Things (IOT) is no longer fictional, experts believe that by 2020 billions of devices and services are predicted to be attached to the Internet of Things. Introduction of IOT has modified the purpose of internet from human controlled computers toward the development of self-running autonomous devices. Smart homes, smart cities, wearables, health care, agriculture, transportation, smart metering, industrial machines and industrial automation are just a few illustrations of the different areas for utilization that are driving the growth of new business models. [1]

The concept of IOT is to incarnate the idea of everything connected. This vision is embraced by a different market such as industrial machinery, healthcare, autonomous vehicles, smart meters, among many others. No single technology is capable of addressing all the IoT use cases [2]. Inside this versatility, Low Power Wide-Area Networks (LPWAN) gained the central focus for IoT. LPWAN is composed of a set of various technologies that can use licensed or unlicensed spectrum and include proprietary or open standard options. Because of low-power, low-speed and low-cost connectivity with wide-range coverage to IoT applications it is progressively gaining popularity in industrial and research communities. It is made up of different technologies using licensed or unlicensed spectrum including proprietary or open standard options. For example, SigFox, LoRaWAN, and Ingenu operate on the unlicensed spectrum whereas Long Term Evolution category M1(LTE-M) and Narrowband IoT (NB-IoT) use the licensed LTE spectrum.[3]

The standardization of IoT and the relevant support structure has been rolled out fairly recently. Earlier in IoT the standardization of LTE-M and NB IoT were not used widely mainly due to cost issues. Emerging IoT market quickly needed a response and thus, Third Generation Partnership Project(3GPP) started a feasibility survey on cellular IoT connectivity. As a result, both LTE-M and NB-IoT are the latest additions to the cellular technology family brought out in the 3GPP Release 13 to deliver wide-area coverage for IoT. Especially NB-IoT supports the LTE specification and reuses the various technical components. It was officially introduced in mid-2016 [2]. The main goal of this advancement is to be able to handle many connected devices and to widen the battery-operated devices' lifetime by using very aggressive sleep algorithms. The features offered by NB-

IoT will allow mobile operators to deliver new network services, adapted to a new User Equipment (UE) profile [4]. It acquires basic functionalities from the LTE system, while it operates in a narrow band. It embraces a recent LTE specification, along with numerologies, downlink orthogonal frequency-division multiple-access (OFDMA), uplink single-carrier frequency-division multiple-access (SC-FDMA), channel coding, rate matching interleaving, etc. [5].

NB-IoT can be spread in three different operation models that is in band, guard-band, and stand-alone. In the in-band operation mode, one or more LTE Physical Resource Blocks (PRBs) are reserved for NB-IoT. The total eNB (i.e. base-station in 3GPP terminology) power is shared among LTE and NB-IoT with the possibility to use power spectral density (PSD) enhancing for NB-IoT. The sharing of PRBs between NB-IoT and LTE allows for more efficient use of the spectrum. In addition, although they are two separate systems, they can be supported using the same eNB hardware. In the guard-band operation, NB-IoT will be deployed within the guard-band of an LTE carrier. In the standalone operation, NB-IoT can be used as a replacement of one or more GSM carriers [6]. The three different NB-IoT deployment options are described in Figure 1.

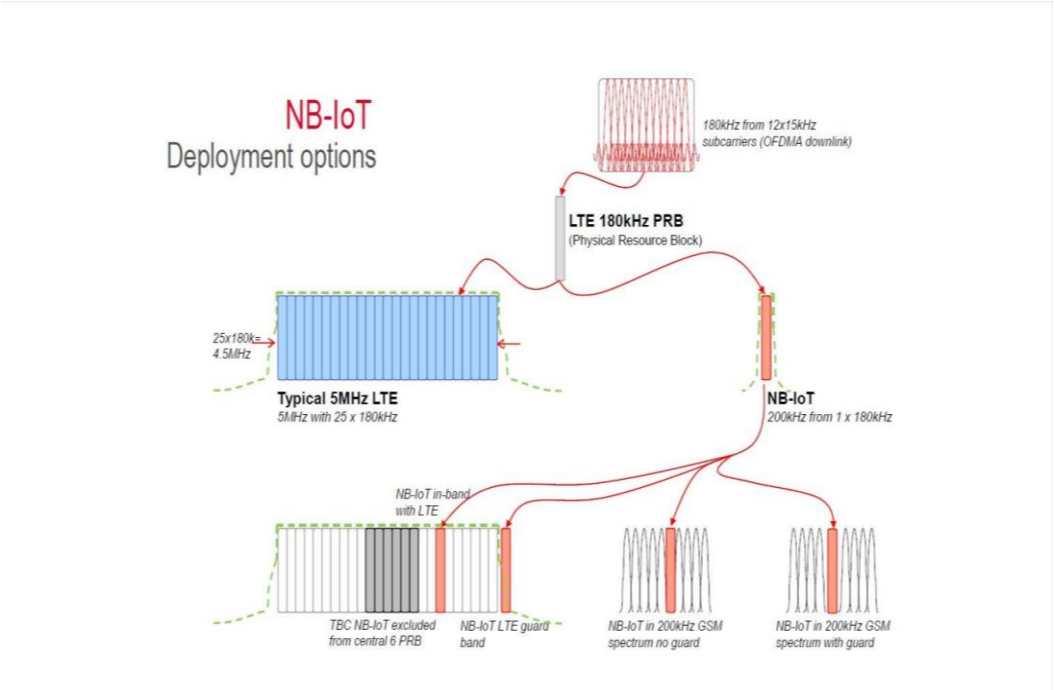

Figure 1. NB-IOT deployment options [7].

Figure 1 describes various options for the deployment of NB-IOT over LTE network. NB-IOT uses similar techniques to LTE. However, the operating frequencies and bandwidth are different.

As NB-IoT adapted the LTE based design it supports most LTE functionalities either with simplification or optimization to provide low-cost, low-power, and low data-rate IoT services. The general characteristics associated with NB-IoT are described in Figure 2.

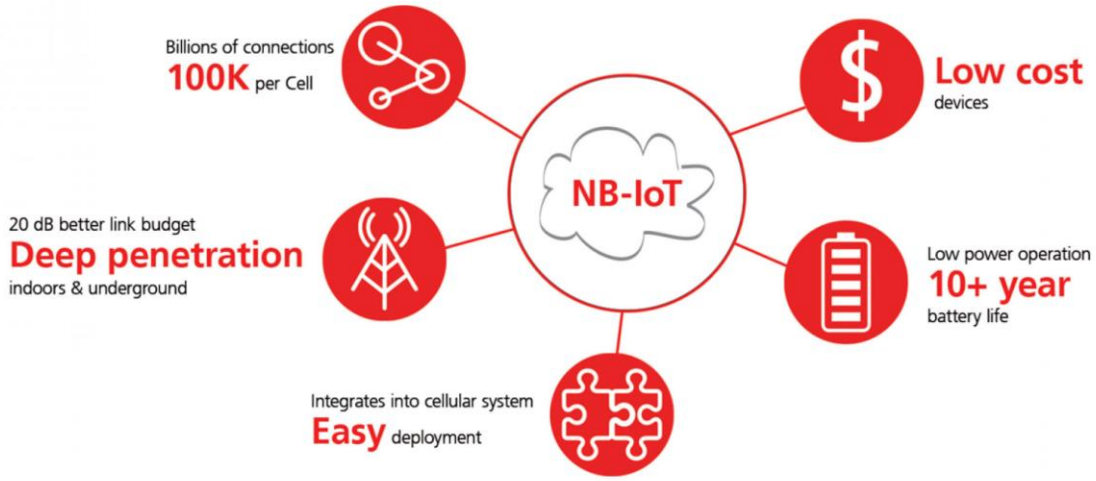

Figure 2. Characteristics of NB-IoT [7].

Figure 2 describes the general characteristics of NB-IoT. NB-IoT deployments must provide low-cost, low power IoT solutions that can support billions of connections and offer deep penetration for a reasonably long period of time.

2 Literature Review

This section contains the review of relevant literature on NB-IoT, BLE and research material on environmental monitoring.

2.1 NB-IoT Design Principles

The rising star, Internet of Thing commonly known as IoT plays a vital role in the information and communication technology (ICT) industry. Cisco predicted that by 2020 there would be 12 billion devices connected [8] whereas Ericson went even further and stated that 18 billion devices would be connected to the internet [9]. Regardless of which of these two made the right estimate such high numbers can hardly be ignored.

Currently there are 7.3 billion mobile cellular subscriptions [9] and the number of connected devices is growing exponentially. Third Generation Partnership Project is the global standardization forum behind the development and maintenance of GSM, Universal Mobile Telecommunication System (UMTS), and Long-Term Evolution (LTE). By 2018 3GPP has a scheme for Release 15 to assure the first delivery of a fifth generation (5G).

Early 2015, the commercialization of Low-Power Wide Area Networks (LPWAN) was taking place rapidly. In France, Spain, Netherland, and the United Kingdom Sigfox was growing out their Ultra Narrowband Modulation networks. The LoRa Alliance was established with a certain intent to accommodate IoT connectivity with wide-area coverage [10]. Abruptly the Alliance collected a notable amount of interest from industry. Until then, Global System for Mobile Communication/ General Packet Radio Service (GSM/GPRS) was the leading cellular technology for providing wide-area IoT. The study on reframing their GSM spectrum to Long-Term-Evolution prompt to non-GMS backward compatible technologies was called a clean-state solution. Neither of the clean-state solutions were specific but the study provided a very reliable ground for the NB-IoT technology which was standardized in 3GPP Release 13. Just after a year of developing the core specification cellular operator vendors started to launch NB-IoT commercially. Now LPWAN technologies provide alternative choices to IoT devices served by GSM/GPRS.

NB-IoT is expected to handle a huge number of devices in the cell with its ultra-low-cost massive Machine-Type Communication design. In order to achieve mMTC demand NB-IoT has to endorse four Key Performance Indicators (KPI) [11]:

- Latency of at most 10 seconds.
- Target coverage of 164 dB maximum coupling loss (MCL).
- UE battery lifetime beyond 10 years, assuming a stored energy capacity of 5 Wh
- Massive connection density of 1,000,000 devices per square km in an urban environment.

2.2 BLE Specifications and Applications

Bluetooth was developed in 1994 by Ericsson, a Swedish telecommunications company to create short range, ad-hoc networks. In 1998, Ericsson formed Bluetooth SIG with IBM, Intel, Nokia and Toshiba to develop and standardize Bluetooth networks and connectivity standards. Bluetooth was designed as a wireless alternative to bulky RS-232 data cables but has grown to be the de-facto standard for short-range communication between mobile and fixed devices. Most smartphones and computers come with Bluetooth support. It is used in day-to-day life to perform data transfer among devices, listening to audio wireless and collecting fitness data from various sensors in wearable devices. [12]

Bluetooth specifications were officially ratified by IEEE as 802.15.1 standard. Bluetooth operates in the 2.4GHz spectrum but it uses adaptive frequency-hopping spread spectrum (AFH) to improve resistance to radio frequencies in the crowded 2.4GHz band and reduce potential interference. The early versions of Bluetooth offered limited range and data transfer speeds. However, with each revision in the Bluetooth standard, data transfer speeds, reliability and security were improved. Transfer speeds increased from 1Mbps in Bluetooth 1.1 and 1.2 to 24Mbps in Bluetooth 3.0 standards. In addition to that, Bluetooth Low Energy (BLE) was added to Bluetooth 4 standards to facilitate low energy data communication with IoT devices that send small packets of data in regular periods. Likewise, Bluetooth 5 comes with improvement frequency hopping and range of connection. Bluetooth 5 also improves on the BLE services with higher bandwidth and greater

range. Likewise, the data security and integrity have also improved with each new specification. BLE specification uses the security mode 4 which enforces service level security with encrypted key exchange. Security mode 4 uses SHA-256 hash and AES CCM cipher for encryption. Security mode 4 was made mandatory from Bluetooth 2.1. In addition to encryption, AFH reduces jamming and interference to preserve the integrity of data and maintain the speed of transfer. Likewise, public key-based pairing and trusted connections further reduce the security risks. The Bluetooth 5 protocol stack is defined in Figure 3 below.[12]

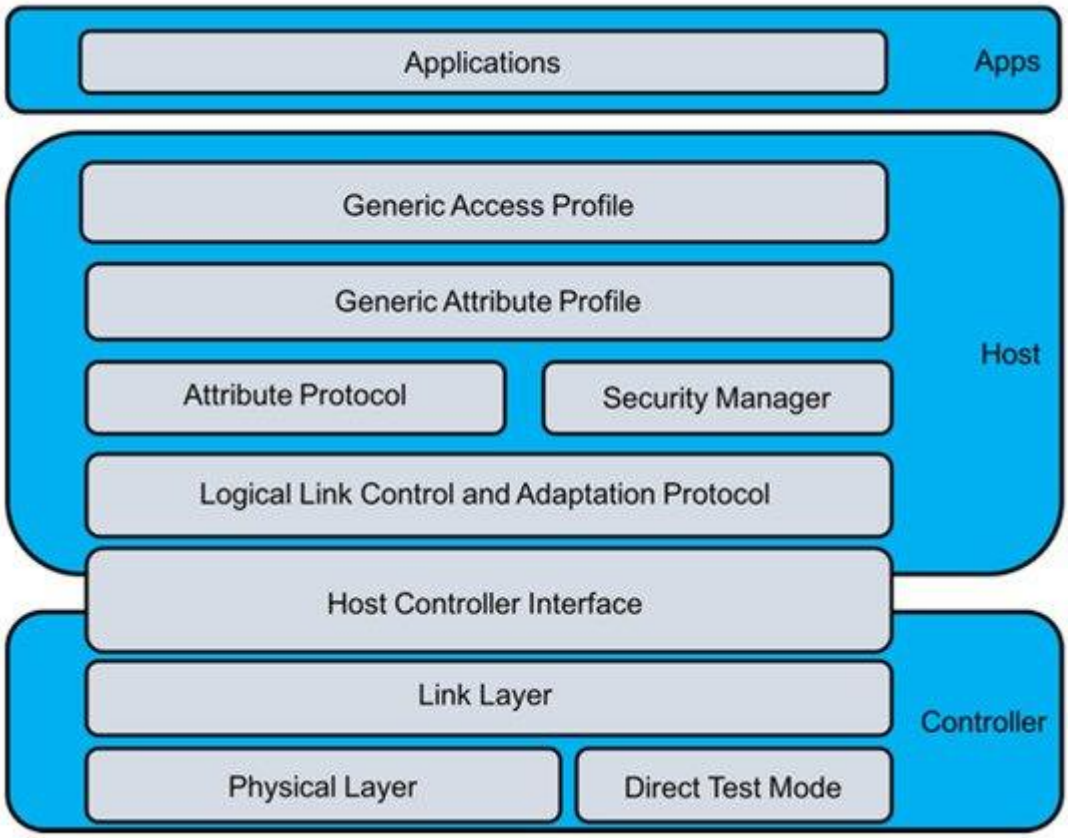

Figure 3. The BLE protocol stack [13]

As described in Figure 3 above, the Bluetooth 4 protocol stack has three distinct layers incorporating various protocols. Each layer is described briefly below:

- Controller Layer: The controller layer is responsible for controlling the radio interface. It is implemented in the hardware containing a Bluetooth radio and a microprocessor. This layer is responsible for controlling the radio in-

terface between two devices and provides functions such as scanning devices and connecting to them. The LE layer is implemented here and is responsible for advertising to and connecting to LE devices and maintaining secure connection with the LE device. The host controller interface standardizes the connection between the host OS and Bluetooth controller governed by the Controller stack.[13]

- Host Layer: Host layer is governed by the device OS. This layer is responsible for data transfer operations. This layer handles the segmentation and reassembly of transfer packets and manages the Quality of Service for higher protocols. In addition to that, the protocols in this layer allow the host OS to determine what services and profile are supported by the connected device. Likewise, host layer ensures the security and integrity of transferred data and pairing of devices.[13]
- Application layer: The application layer provides the applications to interact with the lower layers and provides the interface to the user.[13]

2.3 Environmental Monitoring

Environmental monitoring has become a primary concern for individuals and businesses because of its impact on personal lifestyle and industrial production. In the first decade of the twenty first century, environmental monitoring was considered costly and unnecessary. However, environmental monitoring can provide valuable data that has the potential to contribute in various aspects of individual and communal lives. The environmental data may be used to perform simple tasks such as automated thermostat control and industrial tasks such as production optimization to complicated governmental operations, namely formulation of newer environment protection policies.

Lovett GM. et al. [14] discuss the characteristics of efficient environmental monitoring systems and recommend environmental monitoring as fundamental consideration for environmental science and policymaking. They describe environmental monitoring as measurements of environmental data against time taken at one or multiple locations. In their paper, Lovett et al cite long term measurements of atmospheric CO2 levels at Mauna Loa, Hawaii by Keeling C.D [15] in the 1950s. This measurement led to the proof concept that anthropogenic CO2 emissions are responsible for the rise in global temperature and contribute directly to climate change. This research by Keeling helped to bring forward climate change as one of the most important issues affecting humanity and triggered further research into climate change and data-driven definitions of environmental policies.

Lovett GM et al [14] analyze a set of previous research regarding environmental monitoring and suggest seven-point guideline to create effective and efficient environmental monitoring systems. The guidelines suggest researchers and policymakers to design the monitoring programs around clear and practical questions. Likewise, they recommend the measurements to consider possible changes to data and its impact in the future and include tools for data review and develop adaptable and extendable systems that preserve the integrity of data.[14]

Liu C. et al [16] defined a generic collaborative environmental sensing framework called SCENTS (Sensing Collaboratively in Everyday Networks) to leverage interconnectivity between devices in a given area to collect environmental data with locally significant context. This framework aims to allow nearby devices to collect environmental data efficiently and make data-driven applications less reliant on high-level connection infrastructure. The devices connected to the SCENTS framework periodically scan for nearby devices and sense the capabilities of those devices. The connected devices can communicate with each other to fetch environmental data and create schedules to collect, monitor and send collected data. With this framework, the devices lacking a certain measurement ability can collect the data related to that measurement from other connected devices. In addition to that, nearby devices collecting similar data can pause data collection for a certain amount of time to save energy. Queries made to the SCENTS framework are satisfied by the most relevant observation by a local sensor reading or by context provided by neighboring devices, whichever is the most efficient. They programmed 55 sets of Nordic Thingy 52 sensor kit and placed those in an indoor environment to get relevant measurements to test the viability of the SCENTS framework. They were able to create a flexible and efficient data collection method by encapsulating local and remote sensors to remove the dependence of a sensor on a single application. Liu C. et al were able to pair nearby devices and constantly get data by switching between devices based on charge levels so that one device could keep collecting data when the other needed charging and vice versa.[16]

Bharati PD et al [17] proposed a fog computing platform consisting of Raspberry pi nodes connected to Nordic Thingy 52 to stream environmental data. The fog layer processes environmental data at the nodes themselves and processes the data. They propose us-

ing fog computing to remotely monitor the environment and make context aware decisions by analyzing the collected data and forecast generated ?? with predictive modeling. Their proposal reduces the data sent to the cloud by performing processing at the node themselves. This helps the data collection apparatus to support low latency, real-time decision making and reliability. The proposal by Bharati PD makes it easier for concerned authorities to get required data real-time and make informed decisions on further steps required to solve the problem that could be detected.[17]

3 Device Specifications

The initial objective of this project was to use the newer Nordic Thingy:91 instead of the Nordic Thingy:52. However, the Thingy:91 was not available in the market when this project began. In addition to the sensors and connectivity options present in the Nordic Thingy:52, Nordic Thingy:91 comes with LTE-M, NB-IoT and GPS antenna. In addition to that, it comes bundled with eSIM from iBasis with 10MB data preloaded to provide cloud connectivity out of the box. The presence of eSIM and cellular connectivity eliminates the requirement for fog computing platforms or edge terminals to send acquired data to the cloud [18]. However, Nordic Thingy:91 and the Nordic Thingy:52 are similar to program, support the same platforms and programming languages and provide similar environmental data. In addition to that, since Nordic Thingy:91 is a new product, the volume of documentation is low. Therefore, Nordic Thingy:52 was used instead of the Nordic Thingy:91.

The Nordic Thingy:52 is a very useful development tool for IoT which is marketed for people who want to dive into IoT product prototypes and demos without building hardware or writing firmware from scratch. The Nordic Thingy:52 is built around the nRF52832 Bluetooth 5 from Nordic Semiconductor and is connected to several sensors on the board. All Bluetooth enabled devices such as mobile phones, laptops, tablets, Raspberry Pi can be easily connected to this. The operation of the sensors, e.g. On/Off, sampling rate, etc. can be modified over-the-air via a Bluetooth API which means that it is possible to create demos and prototypes without programming the device. Furthermore, Thingy can be used as a development kit by building own firmware and uploading it onto the board.[19]

The Nordic Thingy:52 also has additional features such as built-in microphone and speaker to support audio which makes it possible to send audio from the device to the app and vice versa. The pre-programed Nordic Thingy:52 can run on any smart phone just by downloading its app which is available for both android and iOS. It also supports IF This Then That (IFTTT) e.g. Philips Hue, Facebook, Gmail, etc. [19]

The key features of Nordic Thingy:52 are listed below.

- Affordable, rapid prototyping and development solution
- No need to develop custom firmware
- Over-the-air configurable
- Bluetooth low energy
- NFC for pairing
- Accelerometer, gyro scope, compass, color, temperature, and air quality sensor
- Microphone
- Speaker
- Built-in Li-ion battery

The software and hardware requirements for each major operating system are listed below.

- Android

  Android OS 6.0 or later with functioning BLE hardware.

- iOS

  Thingy:52 is compatible with iPhone 4+ with iOS 9.0 or above. Likewise, Thingy:52 supports iPad 3rd generation and above. All versions of iPad mini, iPad Air and iPad Pro are supported. The devices also need to have a functional BLE hardware.

- MacOS/OS X

  Thingy:52 requires Bluez 5.41+ is MacOS devices. MacOS support begins with OS X Yosemite. In addition to that, a web-BLE supported browser is required.

- Linux

  Similar to MacOS, Linux devices with functioning BLE hardware require Linux kernel 3.19+ and Bluez 5.41+. In addition to that, a web-BLE supported browser is required.

- Windows

  Windows systems require Windows 10 version 1706 or newer. However, support can be added to systems as old as Windows 7 with Nordic dongle and pc-ble-driver. Windows systems also require web-BLE supported browser to pair to access Nordic Thingy52.

Presently, Google Chrome and Opera are the Bluetooth supported browsers available across all major OS platforms.

The board layout of Nordic Thingy:52 consists of a 6cmX6cm plastic and rubber case with following sensors. The board layout of Nordic Thingy:52 is exhibited in Figure 4 below.

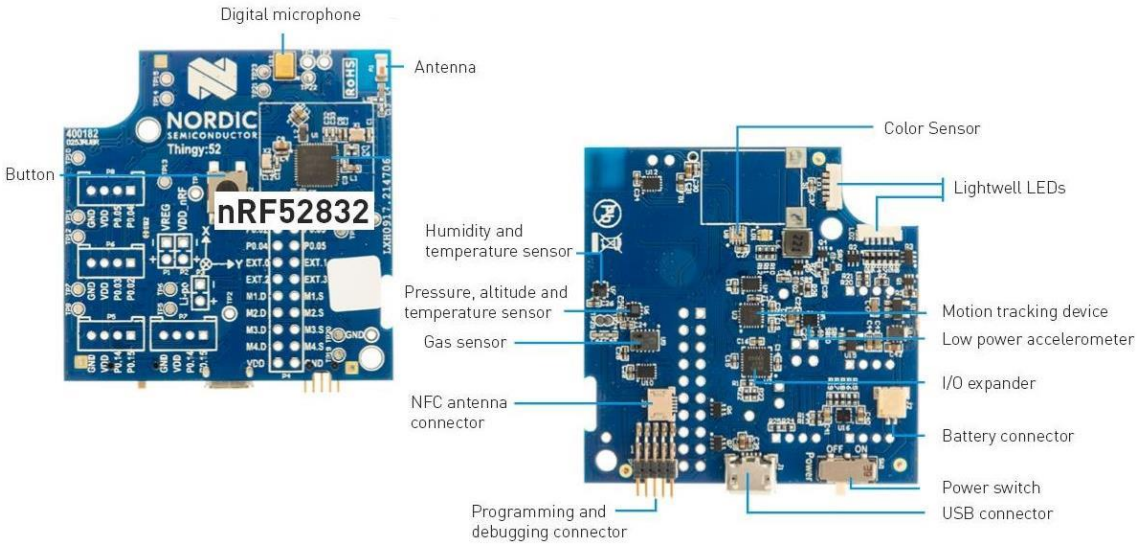

Figure 4. Nordic Thingy:52 board [19]

The sensors present in Nordic Thingy:52 as shown in Figure 4 are listed below.

- 1440mAh battery for a long battery life with micro USB charging
- LPS22HB pressure sensor to measure atmospheric pressure
- LIS2DH12 Accelerometer for orientation data
- HTS221 humidity and temperature sensor
- CCS811 Digital Air Quality Sensor to measure C02 and TVOC levels
- BH1745 Digital Color Sensor
- 9-axis motion sensing (accelerometer, gyroscope and compass) (MPU9250)
- Speaker for playing pre-stored samples or tones
- Microphone to record audio

- Configurable RGB LED and programmable button
- Power switch
- 64MHz Cortex-M4F microprocessor
- BLE 2Mbps radio transceiver

This is an ultra-low-power high performance three-axis linear accelerometer with a digital 12C/SPI serial interface. It provides three different operating modes: high-resolution mode, normal mode and low-power mode.[19]

The Nordic Thingy:52 firmware is built on top of nRF5 SDK v13.1.0 and makes use of its SDKs for peripheral devices, SoftDevice and HAL. The architecture of Thingy:52 firmware is described in Figure 5.

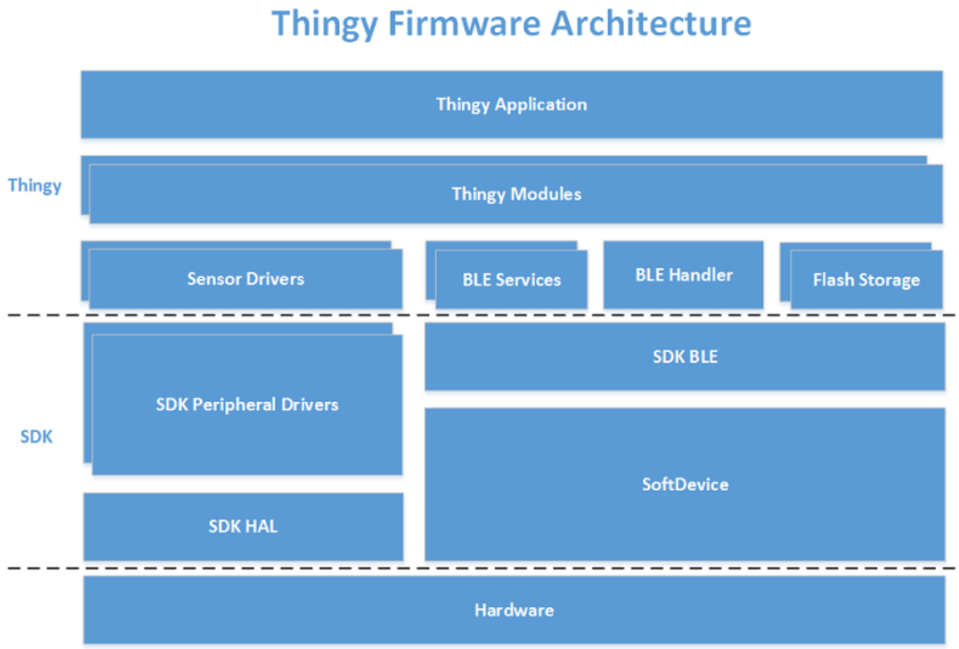

Figure 5. Thingy 52 firmware architecture [20]

Figure 5 describes the firmware architecture of Nordic Thingy:52. Thingy adds following features on top of nRF5 SDK.

- Sensor drivers to perform sensor operations such as enabling and disabling sensors, changing configuration and reading data
- BLE handler to handle Bluetooth connection and dispatch events

- Flash storage to store configuration parameters
- BLE services to handle custom BLE services of the Nordic Thingy:52
- Thingy Modules to control sensor drivers and corresponding BLE services responsible for storing and handling configuration
- Thingy applications [20].

The Nordic Thingy:52 comes with a companion application for web, Android and iOS which provides environmental data. These applications include various uses of bundled sensors such as step counter and magnetic compass. In addition to that, the application can be used to program the Thingy:52 to schedule tasks or trigger operations using IFFT. Finally, the application can be used to update the Thingy:52 firmware when a new version is released.

Nordic Thingy:52 connects to all major operating systems using Bluetooth 5. It can be used to develop native applications for Android and iOS using Java and Swift respectively. Likewise, JavaScript can be used to develop cross-platform applications and Python for web applications or data analysis operations. JavaScript application with React was chosen for this project because it was the easiest and fastest way to show proof of concept. In addition to that, the potential of extending the application and exporting it to support mobile platforms in addition to web application and possible cloud analytics played a part in choice of technology.

4 Application Objective and Target Audience

It is evident from the review of relevant literature that there are numerous applications for environmental monitoring has. The monitoring of environmental data has helped in scientific discoveries and brought about an awareness about global warming and climate change. In addition to aiding in scientific discovery, monitoring of environmental data can also be used for day-to-day activities such as monitoring and optimizing a production environment for higher production. Likewise, environmental data can also be used for formulating new policies. From those observations, a conclusion can be drawn that environmental monitoring is a growing industry and has implications across a variety of areas. Therefore, based on the observations and the review of relevant literature, an environmental monitoring system was developed using the Nordic Thingy:52.

The primary objective of the application is to gather real time environmental data from Nordic Thingy 52 to provide remote monitoring. The data collected by the sensor kit is processes locally and presented to the user as a live graph. This allows the end users to remotely monitor an area and make relevant decisions based on the environmental readings.

The target audience for this application is wide and varied. The application can be used at homes to monitor indoor environment remotely and get real time information about potential accidents such as fire. In addition to that, the application can be used by industries to monitor production environments and optimize the environment variables for maximum production. Likewise, it can be used by cities to start the drive towards smart cities. Furthermore, cities and local bodies can use the real time data to warn city dwellers of hazardous areas, thus contributing to better public health.

5 Core Technologies

This chapter provides a brief insight into the tools and techniques used during the development of the application.

5.1 Programming Languages

The environmental monitoring system described in this paper was developed primarily as a web application serving real time data. The application was developed in end-to-end JavaScript. JSX was used to define react components. JavaScript was standardized in 1996. It is a weakly-typed, multi-paradigm scripting language. The application code is written in ES6 standard. JSX is an extension of JavaScript to incorporate XML like elements and provide an integrated markup syntax.

5.2 Libraries and Frameworks

This section lists and describes the JavaScript libraries and frameworks used in the project in its present state.

5.2.1 Node.js

Node.js is the de-facto industry standard for writing server-side JavaScript code. It is a single-threaded, asynchronous and event-driven JavaScript runtime environment built on Chrome V8 engine. It provides a set of modules that can be used to create webservers and networking tools. Node.js is extendable and the required functionality can be extended by adding third party packages to the project by using npm. Node.js is open-source and supports all major OS distributions. Node.js adapts non-blocking I/O i.e. the commands in Node.js can run concurrently. Node.js puts a request thread to sleep until the desired response is received, waking it up only to deliver the response so that other requests can be received and served in the meantime. This makes Node.js fast, efficient and easily scalable.

Likewise, since Node.js natively supports JavaScript, using it allows developers to deliver end-to-end JavaScript applications. Using a single programming language across the application code makes it easier to debug and maintain. Furthermore, it allows for simpler workflow and tighter integration of server-side and client-side scripts. Finally, Node.js is relatively easy to set up and the availability of various modules and third-party packages reduces the complexity and volume of required code.

In this application, Node.js was used as the web server because of its ease of installation and development. In addition to that, the use of Node.js was a must because the essential libraries for this application, noble-device and thingy52 are available via npm.

5.2.2 Meteor.js

Meteor is a free and open-source isomorphic rapid prototyping platform for developing full-stack JavaScript applications for mobile and web. It is built on Node.js includes a set of Node.js packages and a build tool to provide a thorough platform to develop and deploy JavaScript applications. Meteor integrates MongoDB as a native database system and uses Distributed Data Protocol (DDP) and publish-subscribe pattern to automatically propagate data across the client side without the requirement of state management libraries or additional code from developer. This allows the UI to seamlessly integrate with the API to provide full-stack reactivity with minimal coding effort. Meteor comes bundled with its own templating engine called Blaze, but React templates used in this application are written in JSX.

5.2.3 thingy52

thingy52 is the official Node.js library for Nordic Thingy 52 developed by Nordic Semiconductors. This library provides the set of tools to connect to and access data from Nordic Thingy 52 over Bluetooth connection. This library uses noble-device and noble libraries to handle BLE connection and data transfer.

5.2.4 noble/noble-device

noble is a central BLE module for Node.js available on multiple operating systems. noble-device is built on noble and is used to abstract BLE devices. noble-device is used in this application to discover and connect to Nordic thingy 52.

5.2.5 React

React is a declarative, component-based library used to build user interface. React components are self-contained and reusable. The components are defined in JSX, which is an extension of the JavaScript syntax to allow HTML-like structuring of UI components. React components can accept data in form of 'props' and render the required data. Likewise, react efficiently keeps up with state changes to update the UI when the underlying data is changed. When the state of a component is changed, only the related component is re-rendered instead of the whole page, thus making it faster and more efficient. In addition to that, since react components are able to manage their own state and update accordingly, React separates the state from the DOM. Furthermore, react is extendable and allows the use of multiple other libraries to accentuate the user experience and functionality of the application.

5.2.6 Material UI

Material UI is a collection of react components designed using the Material Design language developed by Google. Material UI provides isolated and self-supporting react components capable of injecting only the required styles. Styles applied to Material UI components are generated at runtime. This application uses Material UI components to render the live data collected from Nordic thingy 52.

5.2.7 Recharts

Recharts is a data visualization library built on React and D3.js. It is a lightweight library that provides simple and adaptable react components to build charts. It provides native SVG support to render scalable visualizations. Recharts is used in this application to render the live charts.

5.3 Database

This application uses MongoDB to store live readings from Nordic Thingy52. MongoDB is a document-based database that stores data as BSON objects. MongoDB stores data as a JSON like object consisting of key value pairs as opposed to rows in SQL. Likewise, MongoDB supports dynamic and flexible schema, as the documents themselves do not need to have a predefined schema. This makes MongoDB flexible, fast and efficient. This also allows for database fields to hold arrays and more complex data structures, which is not possible when using SQL databases. MongoDB was designed for highly available, scalable and high-volume data storage.

MongoDB was chosen for this application because of its flexibility, high volume capability and support for array storage. In addition to that, MongoDB supports native JSON-like queries in JavaScript, thus making it easier to integrate into the application. Furthermore, its scalable and database queries are simpler, more intuitive and faster.

5.4 Tools and Version Control

This application was built and tested in a Linux environment. WebStorm IDE was used during the development as it has native JavaScript support and provides syntax highlighting and code completion for multiple JavaScript libraries. In addition to that, it can be extended by using plugins to add various features such as linting. Furthermore, it provides an in-built terminal to deploy, debug and monitor the application.

This application is hosted in a private GitHub repository. Git was used as a version control system because of its ease to use, integrity and speed, In addition to that, git also checks packages for vulnerability and lets the developer know if project dependencies need to be updated, therefore making it easier to update the applications when required.

6 Design and Implementation

This chapter provides details of different steps of the development and implementation of the application and the design choices made during the process.

6.1 Connecting to Thingy52

The application connects to Thingy52 by using the thingy52 Nodejs package provided by Nordic Semiconductor. When the application starts, it scans for Bluetooth devices and tries to discover the thingy. Then, when a compatible thingy is discovered, it is connected via BLE and the functions provided by the thingy52 library for activating sensors and reading environmental data are activated. The snippet for discovery and connection is provided below in Listing 1.

```
function onDiscover(thingy) {
  console.log('Discovered: ' + thingy);

  thingy.on('disconnect', function() {
    console.log('Disconnected!');
  });

  thingy.connectAndSetUp(function(error) {
    console.log('Connected! ' + error ? error : '');

    thingy.on('temperatureNotif', onTemperatureData);
    thingy.on('pressureNotif', onPressureData);
    thingy.on('humidityNotif', onHumidityData);
    thingy.on('gasNotif', onGasData);
    thingy.on('colorNotif', onColorData);
    thingy.on('buttonNotif', onButtonChange);

    thingy.temperature_interval_set(5000, function(error) {
        if (error) {
            console.log('Temperature sensor configure! ' + error);
        }
    });
    thingy.pressure_interval_set(5000, function(error) {
        if (error) {
            console.log('Pressure sensor configure! ' + error);
        }
    });
    thingy.humidity_interval_set(5000, function(error) {
        if (error) {
            console.log('Humidity sensor configure! ' + error);
        }
    });
    thingy.color_interval_set(5000, function(error) {
        if (error) {
            console.log('Color sensor configure! ' + error);
        }
```

```
    });
    thingy.gas_mode_set(1, function(error) {
        if (error) {
            console.log('Gas sensor configure! ' + error);
        }
    });

    enabled = true;

    thingy.temperature_enable(function(error) {
        console.log('Temperature sensor started! ' + ((error) ? error : ''));
    });
    thingy.pressure_enable(function(error) {
        console.log('Pressure sensor started! ' + ((error) ? error : ''));
    });
    thingy.humidity_enable(function(error) {
        console.log('Humidity sensor started! ' + ((error) ? error : ''));
    });
    thingy.color_enable(function(error) {
        console.log('Color sensor started! ' + ((error) ? error : ''));
    });
    thingy.gas_enable(function(error) {
        console.log('Gas sensor started! ' + ((error) ? error : ''));
    });
    thingy.button_enable(function(error) {
        console.log('Button started! ' + ((error) ? error : ''));
    });
  });
}

Thingy.discover(onDiscover);
```

Listing 1. Thingy52 connection code

The snippet defined in Listing 1 above describes the discovery of and connection to thingy52. Once the thingy is found and connected, the sensors are turned on and the interval to collect data is set. The button in the thingy is programmed to pause the data collection from the sensors.

6.2 Data Collection

The environmental data received from Nordic Thingy 52 is sent to individual MongoDB collections for each sensor reading. The time of the reading and the reading itself are the only aspects of the data sent to the server. The example data is presented in the snippet below in Listing 2.

```
data = {
      time: "MMMM DD, h:mm:ss"
      readings:{
            reading1: value,
            ….
            readingn:value
      }
 }
```

Listing 2. General data model for collected sensor readings

The snippet detailed in Listing 2 above shows the general schema used to store the data. The data is stored in individual snippets as opposed to single collection for the ease of access and isolation in potential predictive modeling and automation applications. The data collection utilizes the functions provided in the thingy52 library. The snippet of the function is provided below in Listing 3.

```
function onTemperatureData(temperature) {
    let data={
        'time':moment().format('MMMM DD, HH:MM:SS'),
        'temperature':temperature
    }
    insertToCollection('temperature', data)
}
```

Listing 3. Storing collected data to the database

The snippet described in Listing 3 describes how the temperature data is collected. When the temperature sensor provides a reading and the thingy library receives it, the data is parsed into the model defined in Listing 3 above and inserted into the relevant MongoDB collection. The insertToCollection function used to insert data to the MongoDB collection is described in Listing 4.

```
//Definitions for Mongo connection parameters are not presented here
function insertToCollection(col, data){
    client.connect(err => {
        if  (err) console.log(err)
        else {
            const collection = client.db("meteor").collection(col);
            collection.insertOne(data)
            client.close();
         }
     });
}
```

Listing 4. insertToCollection function

The insertToCollection function defined in Listing 4 is used to store all the readings. The function takes two parameters, col is a string that describes the collection to insert data into whereas data is the actual JSON data to be stored. MongoDB automatically generates the keys for all the data entries. The data is collected at pre-defined intervals and updated to the server.

6.3 Visualization

The data is visualized in a React and Material UI frontend by using recharts. This application is developed on Meteor.js, hence, it uses the publish-subscribe model to establish a live connection between the database and the frontend. The received data is then passed as props to the react component which contains the recharts component to render the actual graph. The visualizations are live as they subscribe to the changes in the database that is immediately passed to the react component to alter its state and eventually the graph. The snippet listed below in Listing 5 explains a visualization component.

```
const Temperature=(props) =>{
    const [data,setData]=useState(props)
    useEffect(()=>{
        setData(props.temperature.filter((el,ind)=>{
            return ind>=props.temperature.length-60
        }))
    },[props])
        return (
            <ResponsiveContainer width='100%' aspect={16.0/9.0}>
                <AreaChart
                    data={data}
                    margin={{
                        top: 10, right: 30, left: 0, bottom: 0,
                    }}
                >
                    <XAxis dataKey="time" />
                    <YAxis label={{value:'Temperature', angle:-90, posi-
tion:'insideLeft', offset:10}}/>
                    <Tooltip />
                    <Legend />
                    <Area type="monotone" dataKey="temperature"
stroke="#ac0f16" fill="#ac0f16" />
                </AreaChart>
            </ResponsiveContainer>
        );
    }
export default withTracker(() => {
    return {
     temperature: Temperatures.find({}).fetch(),
    };
  })(Temperature);
```

Listing 5. Visualization component for temperature data

The code snippet listed in Listing 5 describes the temperature live graph. React useEffect() hook is used to look for any changes in the received temperature data, once a change is received, the state of the react component is changed. The changed state is then used to update the chart rendered by the component. Finally, all the visualization components are aggregated into a Material UI TabPanel template as shown in Listing 6 below.

```
<div className={classes.root}>
            <AppBar position="static" color="default">
                <Tabs
                        className={classes.tab}
                        value={value}
                        onChange={(event, newValue) => {
                            setValue(newValue);
                        }}
                        indicatorColor="primary"
                        textColor="primary"
                        variant="fullWidth"
                        aria-label="selection tabs">
                        <Tab label="Temperature" value={1} key={1}/>
                        <Tab label="Gas" value={2} key={2}/>
                        <Tab label="Pressure" value={3} key={3}/>
                        <Tab label="Humidity" value={4} key={4}/>
                    </Tabs>
                <TabPanel value={value} index={1} dir={theme.direction}
key={1}>
                    <Temperature />
                </TabPanel>
                <TabPanel value={value} index={2} dir={theme.direction}
key={2}>
                    <GasChart/>
                </TabPanel>
                <TabPanel value={value} index={3}dir={theme.direction}
key={3}>
                    <Pressure/>
                </TabPanel>
                <TabPanel value={value} index={4} dir={theme.direction}
key={4}>
                    <Humidity/>
                </TabPanel>
            </AppBar>
        </div>
```

Listing 6. The aggregated TabPanel component

The snippet presented above in Listing 6 is the JSX component definition for the aggregated live readings received from the sensors. The final user interface is presented in Figure 6 below:

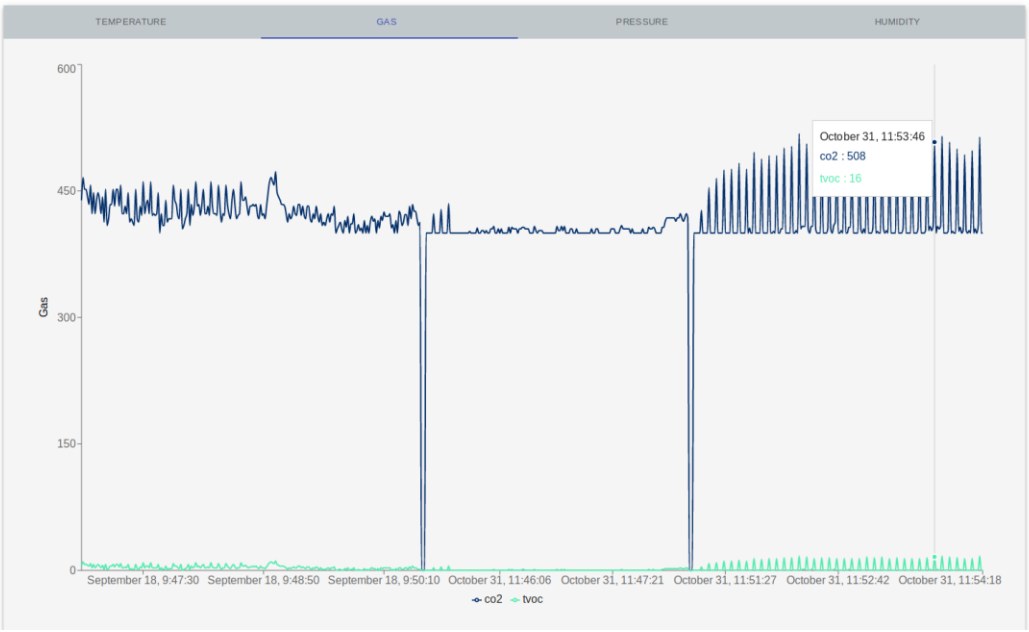

Figure 6. The main UI of the application showing live sensor readings

Figure 6 above shows the final user interface of the application. The end user can access the application via a web browser and monitor the live readings from the thingy sensors. The user can then make informed decisions regarding production environment or home control by reading information from the provided data. The user can switch between tabs to view supported readings.

7 Testing and Results

Environmental monitoring using Nordic Thingy:52 is currently under development. Currently, the application can read the data from Nordic Thingy over Bluetooth and supports remote live monitoring. The data received is successfully stored in MongoDB database. The application was tested in cloud and by using port forwarding to test remote monitoring. The remote testing was successful. However, there is a delay of a few seconds between the thingy reading data and plotting the graph.

8 Ongoing and Proposed Developments

Currently, the only working feature of the application is remote monitoring of the given environment. In the next phase of development, the application will be integrated to the cloud and use multiple sensors to aggregate data. The aggregated data will be used to perform various analytics tasks such as predictive modeling. These modeling tools could be used to perform long-term monitoring of a selected area to calculate the effects of climate change. When the aggregation of data is achieved, the application could be used to perform failsafe monitoring with multiple parallel sensors. Finally, the live readings could be accumulated and connected with other internet enabled devices to facilitate automation in home and production environments.

9 Discussion and Conclusion

In conclusion, Internet of Things (IoT) is a fast-growing area of the technology market. Tens of billions of devices are projected to be connected to the internet in the next few years. IoT enabled devices have a range of application areas ranging from remote control and monitoring to home and industry automation. However, it is important that these IoT enabled devices to be small in size and consume less power. Therefore, to counter the problem of energy usage noble solutions such as BLE, LPWAN and NB-IOT were developed. These implementations provide low bandwidth, highly available and cost-effective connectivity solutions to connect small IoT devices and sensors to the internet and various other services. Environmental monitoring is one of the potential application areas of NB-IoT.

Environmental monitoring has significant challenges and important consequences in a variety of scenarios. In this paper, the author describes the development and usage of a live monitoring application using Nordic Thingy 52 and the potential application areas of said application. From this study, it is evident that environmental monitoring can lead to significant improvements in areas such as climate science and home and industry automation. The research is in this field is limited, however, some of the research has been used in defining climate policies. Furthermore, the use of isolated fog computing edges connected to IoT enabled, low power sensors has been used to collect and process contextual environmental data for remote monitoring and predictive modeling. The usage of fog edges reduces the volume of data and processing at the cloud, thus resulting in more efficient systems.

Therefore, based on observations from the review of relevant literature, a remote environmental monitoring system was developed. The environmental data was collected by Nordic Thingy 52 sensor kit and stored in local and cloud MongoDB databases. The data was presented to the users in a meteor app with React and Material UI frontend. The application provides live readings of selected Thingy 52 sensors.

This document provides the insight into Nordic Thingy 52 kit, NB-IoT with BLE enabled devices and environmental monitoring. In addition to that, it explains the potential applications and enhancement to create environmentally aware home and industry automation.

Reference

1 Keysight Technologies. Narrowband IoT (NB-IoT): Cellular Technology for the Hyperconnected IoT [Internet]. 2017. Available from: http://literature.cdn.keysight.com/litweb/pdf/5992-2360EN.pdf

2 Shin E, Jo G. Structure of NB-IoT NodeB system. International Conference on Information and Communication Technology Convergence. IEEE; 2017. p. 1269-1271.

3 Nair K, Abu-Mahfouz A, Lefophane S. Analysis of the Narrow Band Internet of Things (NB-IoT) Technology. Conference on Information Communications Technology and Society (ICTAS) [Internet]. IEEE; 2019 [cited 4 October 2019]. Available from: http://doi:10.1109/ICTAS.2019.8703630

4 Foni S.et al. Evaluation methodologies for the NB-IOT system: issues and ongoing efforts. AEIT International Annual Conference [Internet]. IEEE; 2017 [cited 4 October 2019]. Available from: http://DOI: 10.23919/AEIT.2017.8240557

5 Dai G, Yu J. Research on NB-IoT background standard development characteristics and the service. Mobile Communications. 2016;40(7):31-36.

6 Ratasuk R. et.al. Overview of narrowband IoT in LTE Rel-13. IEEE Conference on Standards for Communications and Networking (CSCN) [Internet]. IEEE; 2016 [cited 8 October 2019]. p. 1-7. Available from: http://DOI: 10.1109/CSCN.2016.7785170

7 Wu JH. NB-IoT Technical Fundamentals [Internet]. Keysight; 2016. Available from: https://www.keysight.com/up-load/cmc_upload/All/20170612-A4-JianHuaWu-updated.pdf

8 Cisco. The Zettabyte Era: Trends and Analysis [White Paper]. Cisco; 2016

9 Ericsson. Ericsson Mobility Report. Ericsson; November 2016.

10 LoRa Alliance, LoRaWAN R1.0 Open Standard Released for the IoT, 2015. Available from: https://www.lora-alliance.org/kbdetail/Contenttype/ArticleDet/moduleId/583/Aid/23/PR/PR.

11 J. Krause,“Study on scenarios and requirements for next generation access technology,”3GPP TR38.913, Sept.2016.

12 Fafoutis X, Tsimbalo E, Zhao W, Chen H, Mellios E, Harwin W et al. BLE or IEEE 802.15.4: Which Home IoT Communication Solution is more Energy-Efficient? EAI Endorsed Transactions on Internet of Things. 2016;2(5):151713.

13 Lonzetta A, Cope P, Campbell J, Mohd B, Hayajneh T. Security Vulnerabilities in Bluetooth Technology as Used in IoT. Journal of Sensor and Actuator Networks. 2018;7(3):28.

14 Lovett G, Burns D, Driscoll C, Jenkins J, Mitchell M, Rustad L et al. Who needs environmental monitoring? Frontiers in Ecology and the Environment. 2007;5(5):253-260.

15 Keeling C, Bacastow R, Bainbridge A, Ekdahl Jr. C, Guenther P, Waterman L et al. Atmospheric carbon dioxide variations at Mauna Loa Observatory, Hawaii. Tellus. 1976;28(6):538-551.

16 Liu C. SCENTS: Collaborative Sensing in Proximity IoT Networks. IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) [Internet]. IEEE; 2019 [cited 8 November 2019]. Available from: http://DOI: 10.1109/PERCOMW.2019.8730863

17 Bharathi P, Ananthanarayanan V, Bagavathi Sivakumar P. Fog Computing-Based Environmental Monitoring Using Nordic Thingy: 52 and Raspberry Pi. Smart Systems and IoT: Innovations in Computing. 2019;:269-279.

18 Nordic Semiconductor. Nordic Thingy:91 Product Brief. [online] Nordicsemi.com. Available at: https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/Nordic-Thingy-91-PB.pdf?la=en&hash=3A6283D3E6A55836687B3F00251A5C979408EFC5 [Accessed 25 Nov. 2019].

19 Nordic Thingy:52 Product Brief [Internet]. Nordicsemi.com. 2019 [cited 2 August 2019]. Available from: https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/Nordic-Thingy52-product-brief.pdf?la=en&hash=4976338D3BEF6549B2C9470834A2EF0094F785D4

20 Nordic Semiconductor. Nordic Thingy:52 v2.2.0 : Firmware architecture. [online] Nordicsemiconductor.github.io. Available at: https://nordicsemiconductor.github.io/Nordic-Thingy52-FW/documentation/firmware_architecture.html [Accessed 25 Nov. 2019].