IoT- Environmental Monitoring

NAME : Sinthuja.V

ID:AUT962921104710

EMAIL:sinthujasinthuja120@gmail.com

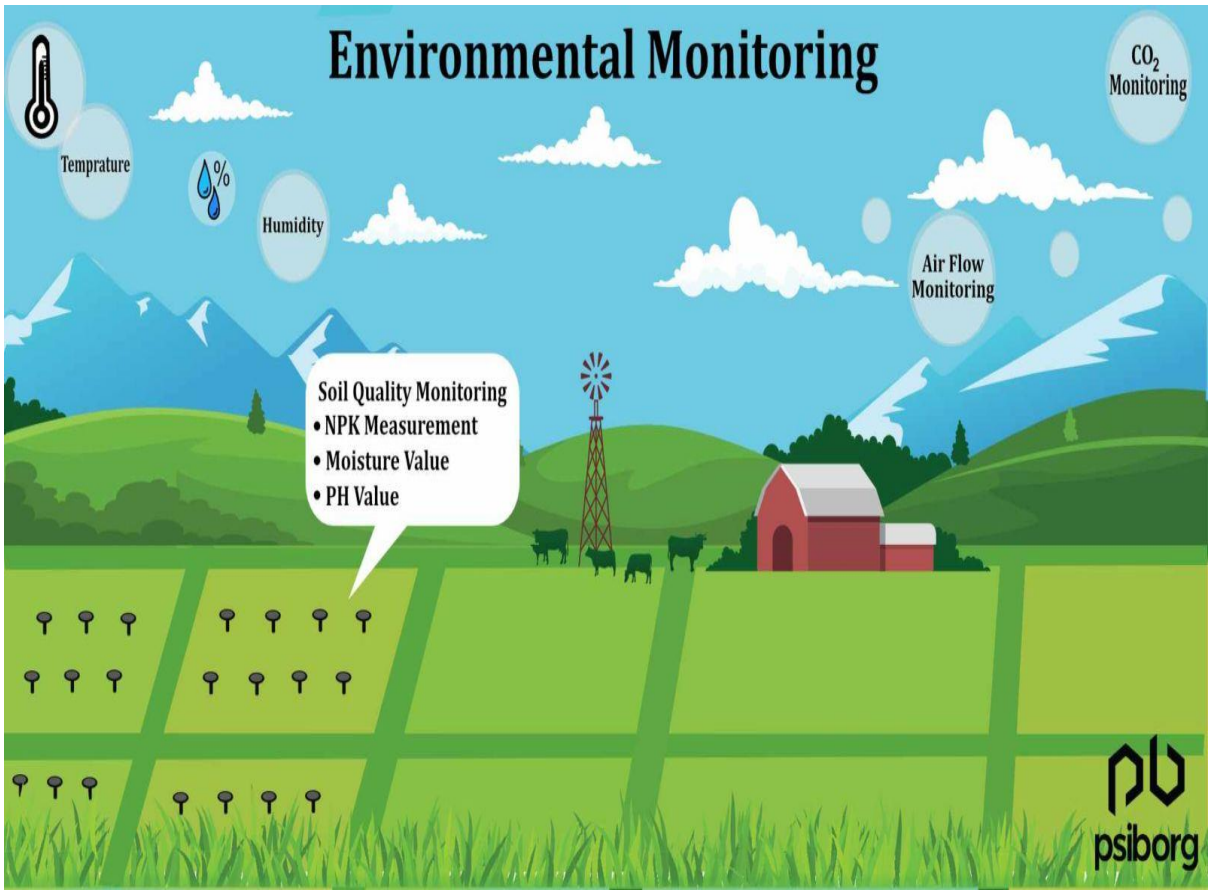


TABLE OF CONTENTS

Environmental monitoring in park system:

1.Introduction:

With the continuous expansion of the total industrial volume, the integration of enterprises has gradually formed a relatively concentrated industrial layout. Industrial centralization has brought advantages in capital, technology and industrial chain, and also put forward higher requirements for environmental protection and safety. However, in recent years, major environmental pollution incidents have occurred frequently, and the life safety of construction workers has not been guaranteed. The root cause is that the supervisory department lacks effective real-time monitoring methods, so that it cannot grasp the pollution situation in time. Traditional human inspections and on-site assessments by professionals have high implementation costs and insufficient coverage of time and space.

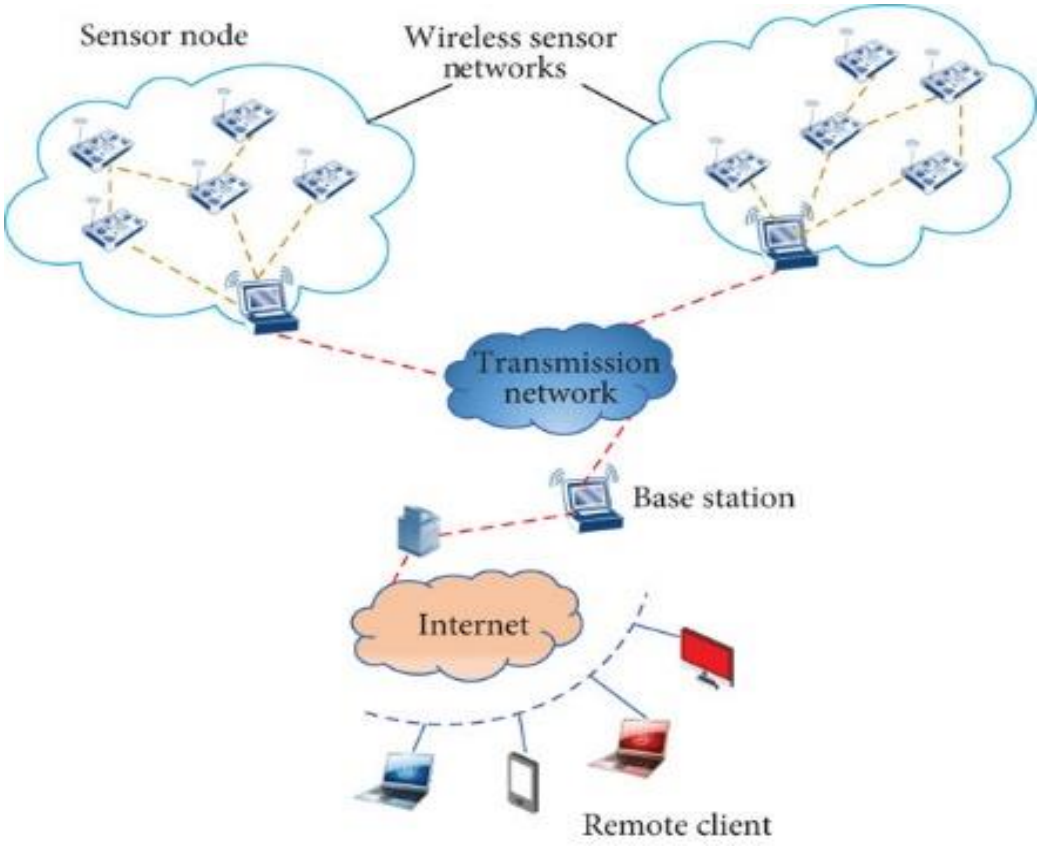
In order to strengthen management, a real-time monitoring system is needed to monitor various parameters of environmental quality in real time and effectively. In the current wireless communication technology market, Bluetooth has a high price, a limited use environment, and a short communication distance; ZigBee's networking method requires strict coordination between nodes and must be deployed at a high density[1]; the mobile cellular network deployed by the operator has characteristics of strong mobility, wide signal range, but the traditional 2G and 4G communication networks are not suitable for application scenarios where small amounts of data are sent and received and power consumption is limited.

In contrast, NB-IoT not only inherits the characteristics of mobile cellular networks, but also has the advantages of wide coverage, large access volume, low power consumption and low cost[2]. In this design, the monitor terminal includes sensors of temperature, humidity, smoke, carbon monoxide, carbon dioxide, TVOC and an embedded system composed of STM32L1 low-power MCU, and the network server is to realize real-time data update and data management functions.

2. Overall Design:

2.1. Overall Structure of the System：

In order to meet the requirements of remote access, scalable capacity, and low power consumption, an NB-IoT-based networking system was designed, which is mainly composed of a monitor terminal, an operator network, and an independently built network server. The monitor terminal is mainly composed of sensors and STM32L151 control chip and BC95. The NB-IoT module transmits the signal to the operator's base station, which is forwarded by the operator to the server, and the data sent by the server is also transmitted to the monitor terminal through the operator. The NB-IoT networking structured in this way realizes multi-point monitoring data and can be transmitted to the server in time. After being obtained, the data is saved in the database and further analysed then displayed on the web page. The overall structure of the system.



2.2. Sensor Selection and Circuit Design:

The temperature and humidity sensor used is DHT11, which is a sensor with calibrated digital signal output. It uses a dedicated digital module acquisition technology and temperature and humidity sensing technology to ensure that the product has extremely high reliability and excellent long-term stability. Its working voltage range is 3V-5.5V, the temperature and humidity ranges measured by the module are 0-50°C and 20-90%RH, respectively. The serial interface in this module can be connected to the IO port of the STM32 microcontroller for data transmission. The smoke and carbon monoxide sensors used are MQ-2 and MQ-7, and their working voltages are both 5V.

The smoke and carbon monoxide concentrations measured by the modules are 100-10000ppm and 10-500ppm, respectively. There is an analog output of the upper limit of 5V in this module, which can be converted to Digital quantity by the MCU of STM32.

The carbon dioxide and TVOC sensor used in the monitor terminal is CCS811, this sensor is based on CCS's unique micro-heating plate technology. Compared with traditional metal oxide gas sensors, CCS811 provides a highly reliable gas sensor solution and a fast test cycle, which significantly reduces the average power consumption. Its working voltage is 3.3V, the carbon dioxide concentration range measured by the module is 400-5000ppm, and the TVOC concentration range is 0-1000ppb. There are clock pin, data pin, wake-up pin, interrupt output pin and reset pin in the module, the sensor's interrupt and wake-up or data transmission can be controlled through the IO port of STM32.

2.3. Controller Solution:

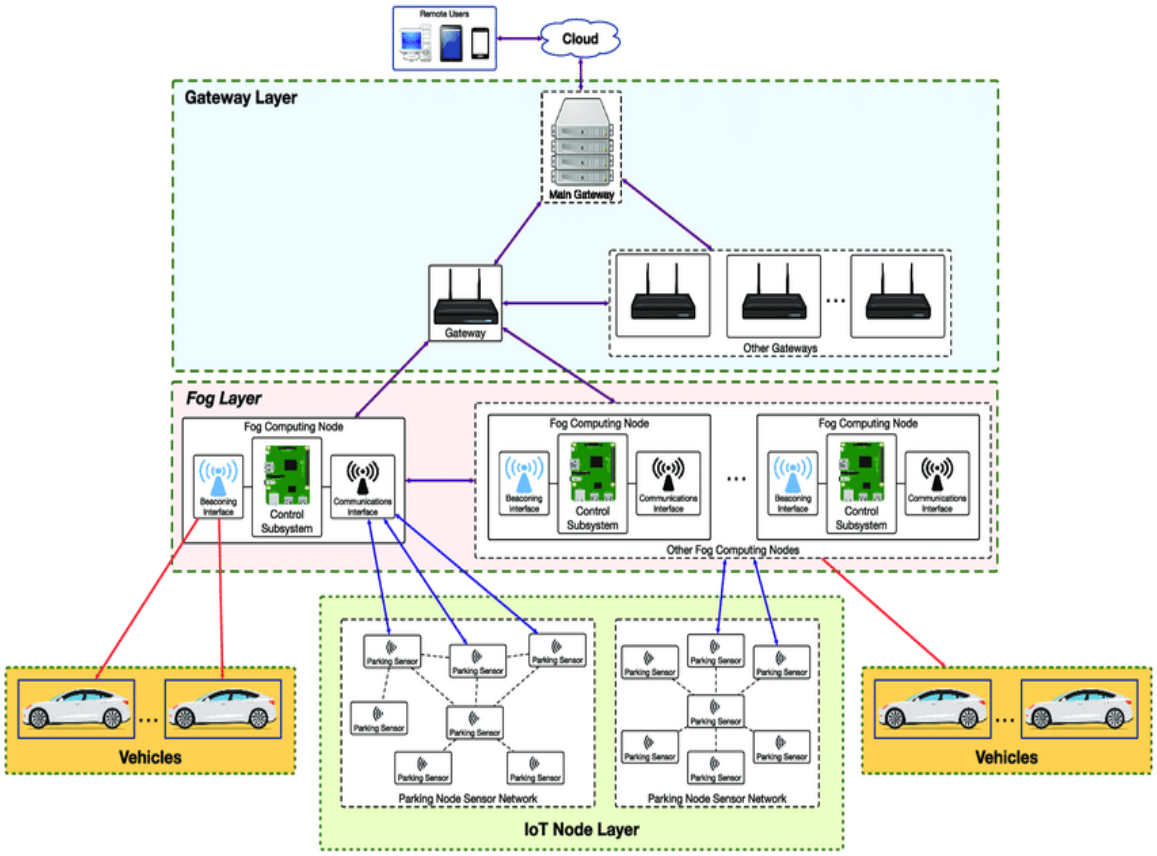
The control circuit of the system adopts STM32L151 as the main controller. The core of the main control chip is ARM Cortex-M3 32-bit CPU, Ultra-low-power 32-bit MCU ARM-based Cortex-M3, 128KB Flash, 16KB SRAM, 4KB EEPROM.

The circuit composition of the minimum system of the MCU includes the power supply stability, external crystal oscillator and so on. Compared with the STM32F1 series of MCUs that use the same Cortex-M3 core, STM32L1 supports more low-power modes, shorter sleep wake-up time, and more sleep wake-up modes, making this series of MCUs able to meet more diverse low-power use scenarios[3]. The system uses STM32L1's STOP mode and Low-Power Sleep mode to plan the workflow of MCU firmware.

2.4. Communication scheme:

NB-IoT stands for Narrow Band Internet of Things. It is built on a cellular network and consumes only about 180 kHz of bandwidth. It can be directly deployed on GSM networks, UMTS networks, or LTE networks to reduce deployment costs and achieve smooth upgrades. Narrowband Internet of Things has a small bandwidth and is not sensitive to delays, but it can transmit small amounts of data stably so that it can be used in environments with weak signals and large amounts of equipment[4-5]. At the same time, it also brings features such as low power consumption, large connection capacity and wide coverage.

BC95 is a high-performance, low-power NB-IoT wireless communication module. It is compatible with M35 modules of GSM/GPRS series in design, adopts easier-to-weld LCC package, has a rich network service protocol stack embedded, and has the characteristics of ultra-low power consumption and ultra-high sensitivity. BC95 supports PSM low power consumption mode. In this mode, the terminal sleeps deeply and does not receive downlink data. Only when the terminal actively sends uplink data can it receive downlink data buffered by the IoT platform.



3. Software design

3.1. Software flow :

The overall software flow mainly includes three steps: system initialization, measuring and processing data, and sending data. When the system is powered on, the system is initialized first, which includes the system working frequency initialization, delay function initialization, BC95 initialization, I2C initialization and so on. After the sensor measures the data, packing the acquired data through a custom communication protocol data format, and then sends the packed data to the server through the NB-IoT module. After the transmission is completed, it will enter the stop mode, and then give a 30-second transmission interval, so that the system is in the low-power mode as much as possible, and only when the module transmits data will it enter normal mode.

3.2. Initialization and Sending data :

The NB-IoT module communicates with the MCU through a serial port, the MCU needs to configure the module to switch on, reset, create Socket, and send UDP data. Before configuring the module, restart the module to facilitate the initialization of the internal program of the module. After restarting, read the CIMI number to make sure that the SIM card is inserted and can be used, after which the module will automatically find the base station, and can manually request the module to register and attach to the operator's network. The flow chart of BC95 initialization is shown in the Fig.2(b). After the measurement is completed, the data needs to be sent to the server immediately. The module supports UDP transmission. First, create a socket and associate with the specified protocol. After the socket is created, data can be sent to the server's fixed port through the UDP protocol. When receiving the data, the server will return an ACK to ensure normal communication, the socket will be closed when the transmission is completed.

4. System testing

Since this chapter mainly introduces the hardware and software implementation of the system, the specific implementation of the server on the network will not be described in detail. The basic process is that when the server receives UDP data, the data is parsed according to the set format and stored in the MySQL database. The original data received by the server and the processed data in the database.

the server receives the data sent by an IP. The beginning of the data is the flag bit and the version number. The following data are separated by'##', which are the values of device ID, smoke, carbon monoxide, carbon dioxide, TVOC, temperature and humidity. In addition, you can see that the corresponding data is stored in the database.

5.Java script

Creating A Node Project

Let's create a simple node application by using `npm init -y`

```
$ npm init -y
```

`npm init` is going to generate a `package.json` file for our project. And the `-y` flag is simply saying yes to all the steps in creating `package.json` file.

Here are the contents of `package.json`:

```
{
  "name": "environment-management",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

You can modify the other fields as per need. For this post, the most important section of the `package.json` file is `scripts`.

Introduction to Environment Variables

An environment variable is a variable whose value is set outside the program, typically through a functionality built into the operating system or microservice. An environment variable is made up of a name/value pair, and any number may be created and available for reference at a point in time.

Structure Of Environment Variables

We create environment variables in name/value pair like:

```
VARIABLE=VALUE
```

Default Configuration Support Using Node.js

In Node.js, you can pass the environment variables when you are running any javascript file using node.

This is the default behavior of how the node binds the variables to the `process.env` object. By default, there are a lot of environment variables defined.

Let's define our first Environment Variable using the command interface.

Let's create a file `app.js` at the root of your project and run it using node.

```
> app.js



console.log('Welcome to configuration management using Node.js');
```

Now, we can run the application using:

```
$ node app.js
```

Output:

Now, let's pass an environment variable via command interface and

```
$ GREETINGS=Hi node app.js
```

Now, node will understand that there is an environment variable `GREETINGS` of which the value is `Hi` and I have to bind this variable with the `process.env` object.

```
console.log(`${process.env.GREETINGS}, Welcome to configuration management using Node.js`);
```

The output will be:

The above method of passing environment variables is fine but it's not scalable.

Environment Management Using `dotenv`

dotenv is one of the most highly used package to manage environment variables.

It's very rich in functionality and supports loading environment files with paths.

Working with `dotenv`

Let's install `dotenv` by simply running

```
$ npm install dotenv
```

After the installation, create a `.env` file at the root of your project. `dotenv` will automatically pick this file and read the environment variables from it. After reading, it will assign all the environment variables to the `process.env` object.

Let's create a `.env` file and write a simple environment variable in it.

```
GREETINGS=Hello World!
```

To use `dotenv` in our node application, import the package at the root of the project or the entry point of your server then call the `config` method of it.

```
console.log(process.env.GREETINGS); // undefined (dotenv is not loaded)



require('dotenv').config(); // load dotenv, all the environment variables are loaded now.



console.log(process.env.GREETINGS); // Hello World!
```

In this way, we can add as many environment variables into the `.env` file as we want. All of them will be parsed by `dotenv`.

Multiple Environments Using `dotenv`

We have looked into how `dotenv` work, and how it loads the environment variables. But this is ideal when you have only one environment to work with.

In real-world projects, we have multiple environments like production, stage, QA and development, etc.

To handle this use case, we can use the `path` option in the `config` method.

Let's create three different files for the production, stage, and development environment. We can add a simple variable of `CONTEXT` in it.

```
$ touch .env.production
```

.evn.production

```
CONTEXT="production"

$ touch .env.stage
```

.evn.stage

```
CONTEXT="stage"

$ touch .env.development
```

.evn.development

```
CONTEXT="development"
```

Now we have environment variable files for every environment, we can load them according to the environment.

Let's load the development environment files. We can use the same traditional command pattern to provide our node app in the context of the development environment.

Let's modify the `script` section of the `package.json` file as:

```
"scripts": {

  "start:dev": "ENVIRONMENT=development node app.js",

  "start:prod": "ENVIRONMENT=production node app.js",
```

```
    "start:stage": "ENVIRONMENT=stage node app.js"

  }
```

We have used the default method of loading environment variables to pass the environment context to our node app.

Now in the `app.js` file, we can create a small function that can return the environment file path based on the `ENVIRONMENT` variable passed in the command line. Note that the `path` option takes the **absolute path of the environment file**, not the relative. That's why we can use the `__dirname` to get the absolute path.

```
function getEnvironmentPath() {

  const environment = process.env.ENVIRONMENT;

  const environmentMap = {

    production: '.env.production',

    stage: '.env.stage',

    development: '.env.development'

  };

  return `${__dirname}/${environmentMap[environment] || '.env'}`;

}
```

Now we can call the `getEnvironmentPath` function within the `config` function of the `dotenv`.

```
require('dotenv').config({ path: getEnvironmentPath() });



console.log('Context: ', process.env.CONTEXT);
```

Let's test our code by running the respective commands.

```
$ npm run start:dev



Context:  development

$ npm run start:prod



Context:  production

$ npm run start:stage



Context:  stage
```

6. Conclusion

Real-time and comprehensive environmental monitoring of industrial parks is indispensable for pollution prevention and safety monitoring, and traditional monitoring systems often have problems of high prices, poor deployment flexibility, and incomplete coverage. In view of the above shortcomings, this paper proposes a set of NB-IoT-based industrial park environment monitoring system. The system mainly includes the hardware side of the acquisition node and the software side of the cloud server. The test results show that the basic monitoring function of the monitoring system is complete and reliable, and it also has the advantages of low power consumption, wide coverage, and convenient interaction. The use of the system to achieve data visualization is more reliable and scientific.