POD - real-time urban pollution monitoring using stationary devices

Adriana Drăghici, Marius Andrei, Dan Tudose
Computer Science Department
University Politehnica Bucharest,
Bucharest, Romania
adriana.draghici@cs.pub.ro, marius.andrei@cti.pub.ro, dan.tudose@cs.pub.ro

***Abstract*—Pollution tracking in urban environments can now be performed through a variety of means, both dynamic and stationary. We envision a pollution data acquisition system that can be made accessible to end users, by including them in the collection process not just presenting the results through web pages or applications. We propose a low-cost device, easily configurable by anyone, which measures gas levels, dust, temperature and noise. The users can deploy them at home, connect them to their local Wi-Fi network and see the pollution data in real-time through a web interface. We address the design challenges of such a sensing device and the iterations that lead to its current hardware and software components. The node can function as part of a system that collects and stores the data and makes it available in real-time to all its users.**

Keywords—air pollution, noise pollution, low-cost device, urban environments

I. Introduction

Pollution represents one of the biggest problems of the modern world, greatly affecting the population's quality of life. In the urban environments, the air pollution is also called smog, a mix of carbon monoxide and nitrogen dioxide, generated by the vehicular emissions. The dust also plays a major role in increasing the air pollution and affects the respiratory system.

The purpose of our work is not to directly offer a solution for diminishing the pollution levels or their prevention, but to raise the awareness of the urban pollution levels and the population's exposure to them. We created a device, called POD, capable of measuring the level of pollution-contributing gases, dust, noise and temperature. Besides sensing, POD performs basic filtering and uploads the data to a service in the cloud.

One of the main goals of our work is to create a community of users willing to share the measured data through their home internet connection to the other citizens.

In order to obtain a competitive device, POD has to provide more or improved features and better pricing than the systems presented in section 2. The main objective is to design and develop an accurate low cost measurement system, capable of directly transmitting the collected data to a server, without the mediation of the data transfers by a mobile application.

Some of POD's requirements in terms of functionality and usability are:

- to incorporate low cost but accurate sensors; creating a device that measures erroneous data is not our goal
- to have a cheap or free internet connection for sending data
- the collected dataset must be easily accessed by the POD's owner and others but the values must not be changed by those
- to support common power sources such as phone chargers
- device configuration should be easily performed, not requiring reprogramming of the main controller. POD targets all users not only those experienced in embedded systems.

Our contributions include the design and implementation of the POD device and the hardware and software iterations needed to support the functional and pricing requirements. We also present the technical challenges and the evaluation of the last iteration of the device.

In Section 2 we will discuss the related projects and sensing devices in relation with the requirements of our device and our overall system policies. In Section 3 we present both hardware and software components and in section 4 the design challenges and how we addressed them. Section 5 presents the results and Section 6 offers conclusions and possible directions for future work.

II. Related Work

Pollution monitoring solutions have been around for a while, but the recent technological trends have given us new ways of implementing them. In addition to the existing vehicular-based solutions or stations placed throughout the cities, we observe now an increased interest in using mobile solutions accessible to all users. We have also seen that large even a high-cost project initiated and sponsored by the local government can be inefficient and also offer a poor interface to its users [1][2]. As researchers and also citizens, we are now concerned with finding cheaper and more community oriented alternatives for monitoring the environment.

The participatory mobile sensing paradigm [3] has gained a lot of popularity during the past five years, making the users active contributors to the sensing process. Currently there are numerous sensing applications for crowdsourcing data about the environment, transportation or community specific issues. In urban environments, pollution monitoring implies both air pollution and noise pollution and mobile solutions usually focus on the latter, based on the smartphones' sensing capabilities. Noisetube [4] offers a complete solution for monitoring the noise levels using mobile phones, has a considerable user-base (tens of thousands of users) and offers free access to its data collection API [5].

For air pollution tracking, the mobile-based systems also employ specialized sensors that perform the data acquisition and transmit it to the phone. The users can access the data through a mobile application. The system proposed by Hasenfratz et al. [6] puts the smartphone in control of the data acquisition, not only of the user interface. The sensing device is connected directly to the phone using a serial interface and the user can start the measurements, observe the results and upload them to the server for further processing. The system aggregates the data from all its users and offers the dataset them publicly on a web page. One of the issues regarding the adoption of this system is the constraint imposed by the application, which requires a phone with a modified Android kernel in order to work. The system currently supports only ozone, temperature and humidity measurements.

CitiSense[7] and CommonSense[8] are two community-driven pollution sensing projects employing smartphones. They also use custom wearable nodes but monitor more parameters than [6] and communicates via Bluetooth with the mobile phone, which can be more convenient for its users.

In our previous work [9] we have implemented a system based on dynamic measurements from sensors placed in cars. We measured just air pollution, specifically the carbon monoxide, various oxides of nitrogen, oxides of sulphur and dust particles. Unlike the current system, the previous one wasn't designed for end users, but for installing the nodes in public transportation vehicles (buses, taxis).

While the combination of handheld sensing devices and smartphones leverages the users mobility to track pollution in multiple locations, it is has constraints in terms of energy consumption (it is battery dependent), size and incentives for the users to carry them. Our sensing node is designed to be independent of mobile applications, and upload data directly to the server. Of course, the system may offer mobile applications for visualizations and account control, but they are not dependent on the nodes. Since the node is designed for home usage it does not rely on batteries and so it offers more sensors and higher sampling rates than the devices discussed above. The disadvantage of not collecting data from all the areas the user goes can be diminished by having a large enough community of users covering many areas of the city.

Notable commercial products offering sensing nodes related to our solutions are Smart Citizen System [10], RESPIRA [11] and Air Quality Egg [12]. They can be incorporated as static sensors in systems that collect pollution data, but they do not fit very well with ours due to pricing or features issues (e.g. they do not measure dust levels). The Smart Citizen System device supports almost all the sensing we are interested in, including the noise level detection but it has a considerable higher price, four times higher than of our device. RESPIRA consists of a sensing board for temperature, humidity, CO and SO_2 with no communication support, requiring an additional board for that purpose. Even with that board, it has an issue with the communication frequency, requiring a gateway device to mediate the communication with a server. It also needs a non-standard power source between 6V and 10V, instead of 5V like our device. Air Quality Egg is similar with RESPIRA but also offers a user-friendly packing. Unfortunately, it is not suitable for novice users, since it requires them to install the Arduino programming platform and reprogram it with the SSID and password of their Wi-Fi Access Point. It also has a higher price, costing six times more than ours (240$).

III. The POD Device

A. Functional requirements

As we presented in the previous section, there are already some devices more or less compatible with our purpose. By developing a new one, we intend to combine all the advantages of the existing products and add additional ones.

We require that POD has a low price and a permanent Wi-Fi connection and to access the server through any wireless router. This is the simplest way, since it does not require additional devices through which to transmit the collected data, such as a smartphone. We chose Wi-Fi in favor of a GSM connection because it does not impose additional costs on the user. Moreover, the use of GSM for data transmission would require components more expensive than the Wi-Fi ones.

As a data collection policy we considered that the data acquisition should be performed continuously from all the sensors. Our communication policy imposes that all the collected values are sent to the server at regular intervals (e.g. 2 minutes). The current storage policy is to save the processed data on the server but we plan to extend it in the future to support temporary local history on devices. To support these policies, the system must offer an analog to digital converter because most of the sensors have an analogical output. Moreover, the main controller must support I2C protocol in

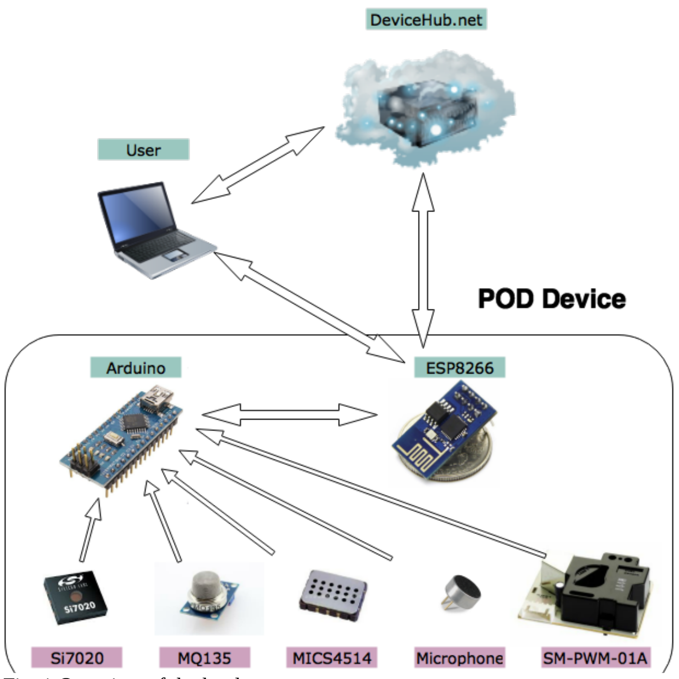

Fig. 1.Overview of the hardware components

order to communicate with digital sensors. Another mechanism needed is the interrupts system, which notifies the microcontroller when an event has happened, for example a pin state change.

In order to be used by anyone, regardless of their technical background, POD must be easy configurable. Consider the following typical user scenario for the first deployment after the user bought a sensing device. The users must get an IDE together with the source code. Then, they must modify the code by writing their home SSID and password and probably also setting a static IP, all of these followed by compiling and uploading the firmware. This is not what an easy configurable device looks like and we do not intend for this scenario to be applicable for our device's deployment. Our goal is to create a device capable of being configured from a personal computer, a tablet or even a smartphone without special cables or adapters.

Measured data must be accessible from all over the Internet. Consider the following scenario: An owner wants to see the temperature from his device but he is at work. To access POD, which is connected to a router, the user needs to know the IP address of that router; usually the IP address is assigned dynamically. First step is to find that address. But POD answers to the request only if the router has a rule set to redirect the request to POD's local IP. Achieving this functionality requires knowledge of computer networks. In order to solve this problem, we considered using a server as a buffer between device and user. DeviceHub.net is an online platform specially designed for IoT, capable of storing data from sensors for as long as the developers request it, being free for the first 30 days. For now, this online platform is perfect for POD, because measured values are stored online, the user can access them from all over the internet and the device memory can be small so no extra costs are involved.

B. *Hardware architecture*

Considering the functional requirements for our device, we will present the components for sensing, data acquisition and communication. Fig. 1 presents how the sensors, the microcontroller and the Wi-Fi module and are connected in order to obtain a functional device.

The sensors we used were chosen based on the following criteria: functionality, communication interface and price. Since most of the gas sensors are a greedy current consumers because they use a permanently powered heaters, energy efficiency was not a criteria when choosing the components.

For measuring temperature and relative humidity, POD uses a low cost pre calibrated sensor, Si7020 [14]. The sensor values can be accessed by the main controller using I2C protocol. Gases are measured using two sensors: MICS4514 - an analogical CO and NO_2 sensor [15] and MQ-135capable of detecting multiple type of gases like CO2, NH3. Those are resistive sensors which means that for each analogical output there is a proportional gas value variable resistor. Measuring CO and NO2 is important because as we mentioned in section 1, and those gases are produced by vehicular emissions. MICS sensor is used by RESPIRA and Air Quality Egg too.

Dust is a another important pollution component and it is measured using SM-PWM-01A, a digital sensor capable to determine small particles (1-2μm) and bigger particles (3-10μm). For every particle, this sensor generates an interrupt maintained as much as the dust particle stays in front of the sensor [16].

To determine the noise level POD uses an electret microphone. But the output signal variation from microphone is so slow that requires an operational amplifier in order to measure wave form. The signal frequency is determined using a Fast Fourier Transform which converts time signal to frequency signal. But the sound level is measured in decibels, so we use the following formula to convert to dB:

$$dB=20\ log_{10}\ A/Ar$$

where *Ar* is reference amplitude and *A* is the measured one.

The main controller a compatible Arduino Nano board.. The reason why we use this board for POD is the price: it is cheaper to use directly an Arduino Nano board instead of buying all those parts and solder them on a PCB. Another advantage is that this board is equipped with a 16MHz crystal oscillator for the AVR controller and an USB-UART adapter, so the firmware can be easily uploaded without the need of a special programming tool.

To access the Internet using Wi-Fi, POD uses an ESP8266 module. This one is very popular in the IOT world. It implements the 802.11 b/g/n protocol and also has the TCP/IP protocol stack integrated [17]. The communication between the Arduino board and the Wi-Fi module is performed through a serial connection. Since the Nano board has just one hardware serial port, which we used for programming and debugging, we simulate a software serial interface and use any of the available pins. We encountered an interconnection problem regarding power levels: the Arduino Nano is a 5V logic level device and ESP8266 is 3.3V logic level device. So in order to communicate and not to damage the ESP, the routes are level shifted using two resistors and a nMOS transistor. The same level shifters are used for the connection between the Arduino Nano and the Si7020 sensor.

C. Software description

This subsection presents the firmware architecture of POD and the manner in which all the functionality policies are implemented.

Arduino supplies a user friendly environment for using the board's peripherals like GPIO, ADC, Serial communication and also provides many open source libraries. By including the compatibility with Arduino in our device's design, any interested user can develop her own mechanism for data acquisition and data uploading to DeviceHub or to any other server. Arduino generated code source is run on an 8 bit AVR architecture microcontroller uploaded using a bootloader. In POD's current firmware implementation, we chose to use directly the standard C library, compatible with Arduino's booloader. In this way our code is more efficient since we could handle the microcontroller's IO and internal peripheral's registers directly.

The communication between Arduino and Wi-Fi module on any of the GPIO pins is possible using the SoftSerial library. Using this library, an Arduino board that supports only one hardware UART interface can be connected to multiple serial devices. For its communication protocol, this connection uses AT commands. For example, when Arduino sends to Wi-Fi module "AT" string, the module responds with "OK". Any other command starts with "AT+" followed by the command name and its arguments. Thus, Arduino must also maintain in memory all the strings needed for server communication, users, passwords and sensors data. When a command is sent, the microcontroller starts a timer, set with the interval needed for waiting the response. In case of timeout, the response is considered unsuccessful.

The board's memory consists of 32 kBytes of Flash, 2kBytes of RAM and 1 kByte of EEPROM [18]. During development, we observed that the RAM size was not enough to keep all the strings and variables. Without attaching any external memory, POD uses the remaining program memory space to store all the strings it needs. The strings declared in the Flash memory cannot be accessed directly so when a PROGMEM declared string is required, it is copied in a general buffer and used from the buffer. For each sensor we use array in which are stored the last achieved values. When POD sends data to server, it sends the average of values stored in buffers. This same action is taken when POD needs to respond to a local user access. It will show all the current measurements in a web page.

Considering the memory constraints, POD is unable to maintain a local archive of data, it can store the data of the last couple of minutes. Currently the system relies on a reliable connection with the server for continuous data transfers, but we consider adding an extra memory in the future. Although it increases the price, it will guarantee no data loses due to problems with the Wi-Fi connectivity. Moreover, with this storage support, the user can have the option to just use the device in local mode, not transmitting the data to the server. The user would not be able to contribute to the community but it is a viable user-scenario for those very concerned with privacy.

Since we considered that the user can change parameters for this device without modifying the source code, POD uses its EEPROM memory for saving Access Point credentials for the Internet connection and the DeviceHub account information. Also, to have an accurate location for device, the user can set the POD's geographic coordinates. This location is saved locally, in the EEPROM memory and also sent to the server. In order to set those parameters, the user must access POD from any browser using its IP address. If the SSID and password stored in memory are wrong, our device creates its own wireless network that user can connect to. After setting the values, the user must restart POD by clicking the web page button "Restart device". Internally, the firmware activates the microcontroller's watchdog and the device restarts, trying to connect to the new Access Point. For easily accessing POD, the user can set a static IP because the mobile DNS service is not implemented yet.

The program installed on the device consists of a loop with a two minutes interval dedicated for data acquisition, followed by an uploading to the server procedure. While the board is measuring values, POD's page can be accessed. This page displays the current measured sensor values and a link to the settings web page. A timer handles an interrupt every millisecond, responsible for data acquisition. When a second passes, it starts an analog conversion for one gas sensor, chosen in a circular order. At every 5 seconds it saves the number of dust particles. When 10 seconds pass, we perform a new temperature and relative humidity measurement. Those values are stored in a circular buffer, one for each sensor and an average is sent to server. Between these intervals POD performs analog conversion for sound measuring applying an Fast Fourier Transform whenever a sound buffer is fill. This feature is still under development and requires more extensive testing.

When it boots up, the device initializes the peripherals end checks if the EEPROM SSID has changes. If so, Arduino sends the new credentials to the Wi-Fi module and waits for the connection to be established. If the settings are the same, the microcontroller just waits for the Wi-Fi connection, because the module is capable to auto connect to the last SSID. If the connection is successful the module starts to work in client function. This means that the device measures data, sends it to server and responds to local requests. If the connection is not successful the device loads the server mode in which creates a Wi-Fi network designed just for configuration. The user must connect to this network and must access the web page located at "192.168.4.1" IP address. This loads directly the settings webpage. After restart the module follow the same procedures.

IV. Design Challenges and Evaluation

When designing a sensing device with cost-restrictions and supporting more sensors than the existing ones discussed in section 2, we came across several hardware and software issues. In this section we will present the iterations our device went through, more specifically the components we tried to integrate and the strategies we adopted for implementing the functionalities

The first design of POD was even cheaper and costing about 30-33$. In this first iteration, the Wi-Fi module ESP8266, which also consists of a microcontroller, was in charge of both the data acquisition and communication. It was also compatible with the Arduino libraries and we could use them for managing the GPIO pins and performing analog-to-digital conversions while preserving its main functionality of a Wi-Fi module. Since it has a single ADC channel we used a multiplexer to iterate through the analog sensors. The problems with this solution were twofold. First, the ADC reference voltage was just 1V and dividing all analog outputs from 5V to 1V results in a lot of noise and precision loss. The second problem concerned the interrupts. Usually, microcontrollers have hardware support for interrupts that signal the end of an analog-to-digital conversion. ESP8266 on the other hand, had only software interrupts support for some of the pins. Due to this fact we had difficulties handling dust sensor interrupts, which did not work all the time. On the other hand, temperature and humidity measurement was functional and the server sockets handling was able to keep more than 5 clients at once.

The problems with the analog-to-digital converter provided by the Wi-Fi module motivated us to change our design. We wanted to eliminate the constraints regarding the single channel, the low resolution and the lack of interrupts. In our second iteration we added an external I2C analogical module with the same precision but a higher reference voltage. Due to price constraints we used a cheap one, less complex, which had only two channels. We used one channel for sound measurement and the other one to iterate through other sensors by using the same multiplexer.

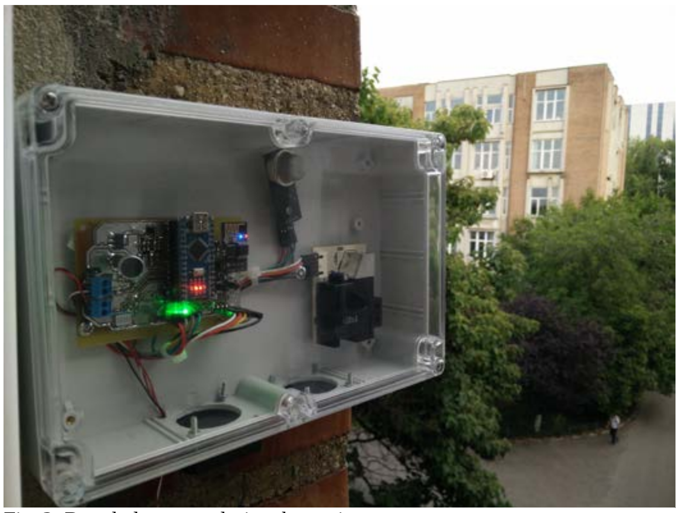
Fig. 2. Board placement during the testing process

In the second revision of POD we encountered problems with the Wi-Fi module's EEPROM memory, which caused the board to reset randomly In the third and current iteration we

added another microcontroller in charge of sensor measurements with a negligible impact on the overall price. Adding this microcontroller implied the same costs as using the analog extension and an additional EEPROM memory.

The current version is stable and all the functionalities described in the previous chapters are implemented. During the evaluation we encountered some software problems such as memory management or responding to local area requests but we managed to successfully solve them. One of the most interesting issue was temperature values which were greater with a couple of degrees like the real ones but relative humidity was right. After some tests we realized that the thermal inertia of the PCB was causing the problem. Therefore we moved the sensor to a dedicated board and the values were normal.

During the evaluation process we placed our device into a box with a cooler which pulled the air out in order to make an air flow inside. The box was fixed at the 4th floor on a window, west side so the Sun was heating up the box after 2pm, (as shown in Fig. 3) when the values rises at about the middle of the graph. We can observe the dependency between relative humidity and temperature because the relative humidity represents the water maximum quantity which can be absorbed at the given temperature.

For dust particles, in Fig.4. we can observe two similar graphs. The sensor is capable to measure particles greater than 2.5µm (higher values - PM2.5) and higher than 10 µm (lower values - PM10)   The small particles sensors measure the others too.

In Fig.5, we can observe the CO and NO2 gases variation during an entire day (July 13, 2016). In the first half of the day the value for CO is constant, about 0.7PPM while the NO2 value increases with 0.3PPM. In the second part of the day, the values has an In the second part of the day, the values have an erratic evolution probably because the window was opened and there was an air flow in the room.

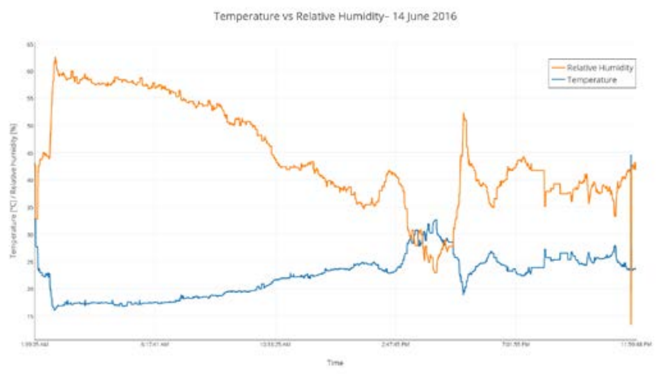

Fig.3.Temperature[*C](blue) vs Relative Humidity[%] (orange) June 14, 2016

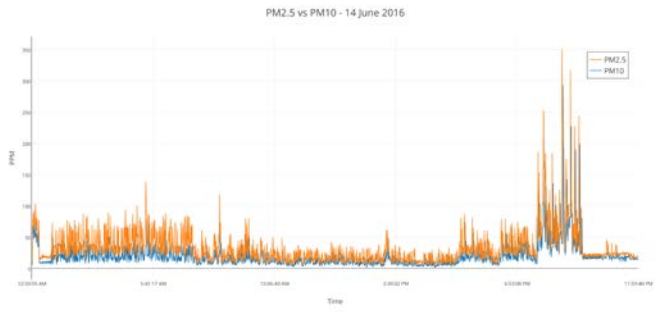

Fig. 4. Dust: PM10(blue) vs PM2.5(orange) [PPM] June 14, 2016

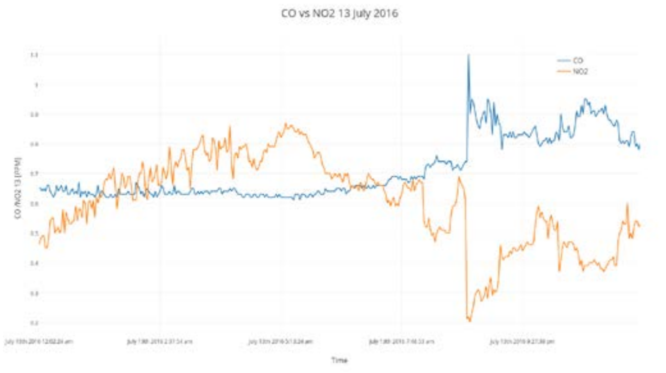

Fig. 5. Gases: CO(blue) vs NO2 (orange)[PPM] July 13, 2016

V. Conclusions

In this paper we presented the key features of the pollution monitoring device we designed and implemented. This device, called POD, is addressed to all citizens interested in obtaining information about air and noise pollution in the areas they live. With a large-enough user-base our web-based system can show real-time pollution information covering many urban areas.

The challenge of this project was to balance the tradeoff between the costs and functionalities. We managed to obtain a device cheaper than the ones available on the market and with a extra features. We started by using a popular device in the Internet-of-Things field, with a cost of about 30$ and we added some extra hardware getting an approximate production price around 40$. We successfully managed to use all desired sensors and to respect our usability policy. The setup of the device is extremely easy and requires no embedded systems or programming knowledge. During our evaluation process no auto restart or system freeze was observed. By presenting the design iterations and the issues we overcome, we hope our work can help other researchers in designing their own solutions using accessible and simple hardware components.

One of the functionalities we plan to add is a mobile DNS service which will ease the device's discovery in the local network. We also consider adding a SD card holder and support for local storage. This will ensure that the data is not lost when there are problems with the user's Wi-Fi connection.

We plan to expand the project with an online platform accessible to the general public in which to present the measured data, updated in real time. Our goal is to create an infrastructure for measuring pollution in Bucharest, Romania, based on these devices.

References

[1] Official website made available by the Romanian National Environmental Protection Agency on pollution reports. (Romanian). Available online at: http://www.calitateaer.ro. Last accessed on 14 July 2016.

[2] Air Quality in Bucharest and its impact on human health. (Calitatea aerului în București. Efecte asupra sănătății) (Romanian). Available online at: http://www.ecopolis.org.ro/media/files/studiu_calitatea_aerului_bucuresti.pdf. Last accessed on 14 July 2016.

[3] Lane, N. D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., & Campbell, A. T. (2010). "A survey of mobile phone sensing". *IEEE Communications magazine*, *48*(9), 140-150.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[4] D'Hondt, Ellie, Matthias Stevens, and An Jacobs. "Participatory noise mapping works! An evaluation of participatory sensing as an alternative to standard techniques for environmental monitoring." *Pervasive and Mobile Computing* 9.5 (2013): 681-694.

[5] NoiseTube Developers Page. http://noisetube.net/api_overview. Last accessed on 14 July 2016.

[6] Hasenfratz, David, Olga Saukh, Silvan Sturzenegger, and Lothar Thiele. "Participatory air pollution monitoring using smartphones." Mobile Sensing (2012): 1-5.

[7] Zappi, Piero, Elizabeth Bales, Jing Hong Park, William Griswold, and T. Šimunic Rosing. "The citisense air quality monitoring mobile sensor node." In *Proceedings of the 11th ACM/IEEE Conference on Information Processing in Sensor Networks, Beijing, China*. 2012.

[8] Dutta, Prabal et al. "Common sense: participatory urban sensing using a network of handheld air quality monitors." In *Proceedings of the 7th*

ACM conference on embedded networked sensor systems, pp. 349-350. ACM, 2009.

[9] Tudose, Dan Ştefan, Traian Alexandru Pătraşcu, Andrei Voinescu, Răzvan Tătăroiu, and Nicolae Ţăpuş. "Mobile sensors in air pollution measurement." In *Positioning Navigation and Communication (WPNC), 2011 8th Workshop on*, pp. 166-170. IEEE, 2011.

[10] Smart Citizen System board. https://acrobotic.com/featured/sck-00001. Last accessed on 14 July 2016.

[12] RESPIRA sensor. http://old.panstamp.com/announcements/respirasensor. Last accessed on 14 July 2016.

[13] Air Quality Egg http://airqualityegg.com/. Last accessed on 14 July 2016.

[14] Silicon Labs, "" Si7020-A20 I2C Humiditry and Temoperature Sensor" Si7020 Datasheet, 2015, Rev 1.1

[15] e2v technologies (uk), " MiCS-4514 Combined CO and NO2 Sensor", 2008, A1A-MiCS-4514 Version 2

[16] Amphenol Sensors "SMART Sensor SM-PWM-01C Application Notes", Nov. 2015

[17] Espressif Systems IOT Team, "ESP8266EX Datasheet", August 1, 2015

[18] Atmel Corporation, “ATmega48A/PA/88A/PA/168A/PA/328/P” ATmega328P datasheet, 11/2015 [Revised Rev.: Atmel-8271J-AVR-ATmega48A/48PA/88A/88PA/168A/168PA/328/328P-Datasheet_11/2015]