IoT Solution for Smart Cities Pollution Monitoring and the Security Challenges

Cristian Toma 1*, Andrei Alexandru 2, Marius Popa 3 and Alin Zamfirou 4

1 The Bucharest University of Economic Studies, cristian.toma@ie.ase.ro
2 The Bucharest University of Economic Studies, alexandru.andrei196@yahoo.com
3 The Bucharest University of Economic Studies, marius.popa@ie.ase.ro
4 The Bucharest University of Economic Studies, alin.zamfirou@csie.ase.ro

* Correspondence: cristian.toma@ie.ase.ro

Received: date; Accepted: date; Published: date

Abstract: Air pollution is a major factor in global heating and the focus is increasing for solving this problem. Urban communities take the advantage of IT and communications technologies in order to improve the control on environmental emissions and sound pollution. The aim is to mitigate the health threatening risks and to raise the awareness in relation to the effects of air pollution exposure. This paper investigates the key issues of a real-time pollution monitoring system, including the sensors, Internet of Things (IoT) communication protocols, acquisition and transmission of data through communication channels, as well as data security and consistency. The security is a major focus in the proposed IoT solution. All the other components of the system revolve around security. The bill of the materials and the communications protocols necessary for designing, development and deployment of the IoT solution are part of this paper, as well as the security challenges. The paper’s PoC – proof of concept addresses IoT security challenges within the communication channels between the IoT gateways and the cloud infrastructure where data are transmitted to. The security implementations adhere to existing guidelines, best practices and standards ensuring a reliable and robust solution. In addition, the solution is able to interpret and analyze the collected data by using predictive analytics, in order to create pollution maps. Those maps are used to implement real-time countermeasures, such as traffic diversion in a major city, to reduce concentrations of air pollutants by using existing data collected over a year. Once integrated with traffic management systems – cameras monitoring and traffic lights – this solution would reduce the vehicles pollution by dynamically offering alternate routes or even enforcing re-routing when pollution thresholds are reached.

Keywords: IoT – Internet of Things, IoT Edge device security, cybersecurity, sensors, IoT communications protocols, smart-cities.

1. Introduction

The air pollution in major cities may reduce life expectancy with up to 22 months. However, according to a study that was made by the World Health Organization (WHO) in May 2016 and updated in 2018, most of the big cities are facing this issue. More than 80% of urban areas reach air pollution levels well above acceptable values [1] and [2] – “around 7 million people die every year from exposure to fine particles in polluted air that lead to diseases such as stroke, heart disease, lung cancer … and respiratory infections”. The World Health Organization is providing an interactive map based on estimates models which is taking into account fine particulate matter (PM2.5) in micrograms/cube meter.
Therefore, this has become a growing exponentially increasing problem within the recent years and its effects are visible in everyday life.

The data obtained by monitoring the busiest urban areas proves that the pollution impacts not only our own lives but also the lives of the generations to come. Therefore, it is very important to try as much as possible to be careful and control the harmful emissions that we release in the atmosphere.

The IoT solution aims to monitor the pollution level, especially the one present within cities by using several IoT devices and attached sensors. The idea involves having multiple stations placed in several areas of the cities. These stations periodically upload and send data to the IoT cloud. For the proof of the concept and the prototyping the authors have used the development boards such as Nitrogen iMX 6 or Raspberry Pi, but for the final IoT solution, proper calibrated high quality sensors and industrial IoT gateway devices such as HMS Netbiter or equivalent equipment are targeted. The key differentiator between this solution and other solutions is the designed security for IoT Gateway and the communications to the IoT Clouds. The AI - Artificial Intelligence and ontologies and other technologies may be used for better prediction and data analysis.

One purpose of the paper is to collect, analyze and process the data received from the sensors in order to obtain real-time monitoring of the air quality as well as the security and consistency of the collected data. Therefore, it is necessary to design and develop new types of public services such as:

• Determination of areas less polluted at a certain moment in time - parks, markets or any other public area;
• Determination of the most polluted areas, in order to avoid them;
• Triggering specific public actions such as partially roads closing in order to decrease the current pollution level or triggering of the alarms if excessive measurements have been recorded;
• Finding out the least polluted area in real estate.

A city is “smart” if it uses different types of sensors and devices to collect data and provide information that is used to efficiently manage resources. Data can be processed and analyzed in order to monitor and manage the traffic and transport systems, the waste recycling, water supply networks and other public services.

Stations, located in crowded areas of the city, periodically read a number of samples containing data from attached sensors (Figure 1). At a set time, the station processes an average sample to reduce erroneous data and sends it to the cloud platform. The data is transferred using a low-resource protocol.

![Figure 1. IoT Monitoring Solution of the pollution – generic view](image-url)
The main contributions of the authors in this paper are the following:

- Design of a large, scalable architecture and infrastructure for the pollution metrics collection within an IoT smart city solution.
- Design and implementation of the PoC – Proof of Concept of an IoT Architecture for collecting pollution metrics within a smart city using the best practices in software development technologies.
- Analysis of the technologies and platforms which may be used within a pollution monitoring solution.
- Security of the MQTT communication protocol implementation for the collected data, in order to avoid indirect attacks on the data collection process. The system also needs to be resilient to the malicious threats – e.g. cyber terrorism - that could represent high risks once the integration with any traffic control system is made.

The paper is structured in five sections, starting from the sensors data collections, IoT gateway, communications protocols, IoT Cloud to the security challenges and conclusions. The second section presents the existing solutions for pollution monitoring and shows the differences between them.

The third section shows the bill of the materials and the communications protocols necessary for designing, developing and deploying the IoT solution. It also emphasis the architecture of the components, the data flow implementation details and the edge connectivity to the IoT Cloud.

The fourth section presents the security challenges and the PoC approach merits. The last section shows conclusions and future work, as well as results of the pollution monitoring gathered for one year inside a major city (e.g. Bucharest, Romania, European Union in year 2018).

2. Related works

According to [3] air quality assurance is a very big concern for the humanity, especially for the people living in the urban area. There is a custom extensible AQIoT (Air Quality Internet of Things) device platform capable to make measurements and store all collected data in cloud. The components used in the first version of the device cost around 900 US dollars and are based on a Raspberry Pi3 Model B. This version has been tested with six devices in Southampton in the UK. The communication between the devices is made via a LoRaWAN network. The second version of the device - costing around 1000 US dollars – has been developed right after. Another approach is presented in [4], using a Raspberry Pi2 Model B board. Other perspectives on solving similar problems are presented in [5], [6].

LoRaWAN is used also in a Romanian solution named uRADMonitor [7]. It equips the smart cities project with sensors for various pollutants. This solution has been used in Alba Iulia with a number of 15 sensors installed on 15 busses. This novelty of this approach is in this way the sensors work and the platform ability to render maps with the measurements decks and dashboards. Samples of the collected data can be accessed at [8].

Another pragmatic project is RadonAir - developed by the Dositracker start-up. It aims to reduce the concentration of radon in the dwellings and buildings [9]. Another research that analyzes the smart Buildings is [10], where the authors propose a system which focuses on efficient energy usage for data collection.

According to [11], as people spent 90% of their time indoors, home air quality is also very important. The iAir system is used for collecting data about ammonia, carbon monoxide, nitrogen dioxide, propane, butane, methane, hydrogen and ethanol. The iAir provides real-time alerts when the concentration of one of these gazes is excessive. The system uses an ESP8266 as a microcontroller and a MICS-6814 sensor for measurements. The collected data is stored in ThingSpeak which allows data analysis on time series. The micro computing unit - MCU ESP8266 is also part of another solution presented in [12].

The Internet for Things technology is used for the monitoring of both air and water pollution. Another IoT project with SCADA technology for the water quality monitoring is presented in [13].
The security of these systems is an important aspect and several studies, such as [14], focus on its implementation in data integration and air quality monitoring.

The second version of the solution aims to use IoT-NB 5G GSM Network in a hybrid approach with Wi-Fi, SigFox and LoRaWAN and to enhance security by using Java Card secure element in within an industrial IoT Gateway. This uses at least IP 67 because of the outdoor exposure of the station. The development boards such as Raspberry Pi 3 and Nitrogen iMX6 are suitable candidates for the proof of the concepts and fast prototyping but in real solutions they aren’t reliable.

3. IoT Platform for monitoring of the smart cities pollution – IoTP4mSCp Solution – Architecture, implementation, material and methods

The Internet of Things platform for monitoring of the smart cities pollution (IoTP4mSCp) is containing the following components:

- Wired and wireless sensors for various metrics measurements;
- IoT Gateway(s)/Node(s) for the data collections – for the moment, development boards are used for the proof of the concept;
- IoT communication middleware with security features for sending data to IoT Clouds.
- IoT Cloud solution for mathematical, statistics and AI – Artificial Intelligence models for data analytics and data science techniques – the authors are still working on these models.

The following subsections are presenting the details of the implementation. The first focuses on the components configuration - Cloud services, MQTT broker, Node-RED, MongoDB database as well as on the implementation of the platform solution through the gateway application.

3.1. IoTP4mSCp Architecture and Data-Flow

The data flow diagram (shown in 2) presents the logical flow of the data processing and transmission functions through the current architecture.

The architecture (figure 2) has the following components:

- IoT Edge devices
  - IoT wired (I2C, SPI and UART) and wireless sensors (ZigBee / Z-Wave) – used to collect the following data using 5 seconds sample rates:
Temperature (°C), Pressure (hPa), Altitude (m), Relative humidity level (%), Carbon Dioxide (CO2 ppm), Carbon Monoxide (CO ppm), Ammonium (NH4 ppm), Methane (CH4 ppm), Detected Wi-Fi networks and their signal strength in decibels.

- **IoT Smart Objects**
  - IoT Nodes – for the prototype and proof of the concepts it is ok to use ESP8266, but in production industrial/outdoor MCU – micro computing units should be used.
  - IoT Gateways – for the proof of concepts, the development boards (e.g. Raspberry Pi 3 and Nitrogen iMX6) are enough, while in production, real IoT gateways such as HMS Netbiter or Eurotech or equivalents should be used. These are a bit more expensive than the IoT Nodes but have greater computation power and their role is to supervise the IoT edge devices communications and cooperation.

- **IoT Networks / Internet** – the communications to the Internet/IoT Cloud are important and need to be reliable and secure, ensuring at least city coverage. The following network communications may be used: GSM (2G/3G/4G–LTE/5G – IoT-NB), SigFox, LoRaWAN and Wi-Fi.

- **Solution Clouds** – the prototype used two types of the clouds:
  - IaaS/CaaS (Amazon EC2 with containers as Docker and Kubernetes) – a virtual machine with Docker/Kubernetes container where the authors have deployed: the webserver (Apache Tomcat) with RESTful services, the MQTT broker (Mosquitto), node.js 9+ and Node-RED frameworks and the NoSQL database (MongoDB).
  - IoT PaaS dedicated Cloud (such as Google IoT, AWS IoT, IBM Bluemix IoT and Oracle IoT in order to not be depended by any cloud provider. These clouds are used for dedicated IoT time series analytics. The data processed in this section is transferred to either the enterprise systems for Business Intelligence, reports and forms, or to the front-end consumers that allow data management and reporting.

- **Enterprise Systems**
  - BI – Business Intelligence, AI – Artificial Intelligence and data analytics clouds or enterprise systems may be used for providing a strong interpretation of the process data. This part is in progress within the research project of the authors.

- **Front-End devices for management and reporting** - e.g. smart city pollution maps creation.

IoT gateways pollution monitoring stations are MQTT subscribed to the broker topics: /stations and /stations/[identifier station] – where [station_identifier] represents the MAC address of the station network interface card if Wi-Fi/Ethernet/GSM networks are used. In these topics they are waiting for the configuration messages. After initialization, the stations collect data from the sensors at a specific interval and publish an MQTT message to the Mosquitto broker hosted on the AWS EC2 side.

The Node-RED component persists the data through the MQTT protocol. It launches a stream that listens to messages posted on the topic / sessions, takes over their payload, validates it, and then stores it into MongoDB.

The IoT4mSCp Cloud/Backend component provides RESTful services for interacting with the MongoDB database (CRUD operations) and with the MQTT broker. The messages which are published in the above broker topics modify the configuration parameters of a station - enable / disable, interval where data is retrieved / sent.
3.2. Hardware Bill of Materials

To monitor the level of air pollution and air quality, a development board (such as Raspberry Pi or Nitrogen iMX) station records the following data with attached sensors: temperature (°C), atmospheric pressure (hPa), altitude (m), humidity level (%), carbon dioxide level (CO2 ppm), carbon monoxide (CO ppm), methane (CH4 ppm) and ammonium (NH4 ppm). Also, data on Wi-Fi networks identified in the vicinity of the station and the level of their signal are recorded. The components that make up a station are shown in Table 1 - Bill of materials.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Quantity</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT Development board</td>
<td>The board Raspberry Pi 3 Model B with 40 GPIO pins or Nitrogen iMX board.</td>
<td>1</td>
<td><a href="https://www.sparkfun.com/products/13825">https://www.sparkfun.com/products/13825</a></td>
</tr>
<tr>
<td>Power Source</td>
<td>Power source 2.5A and 5V.</td>
<td>1</td>
<td><a href="https://www.adafruit.com/product/1995">https://www.adafruit.com/product/1995</a></td>
</tr>
<tr>
<td>SD CARD 16GB</td>
<td>SD Card which store Raspbian OS or Embedded Linux Ubuntu.</td>
<td></td>
<td><a href="https://www.adafruit.com/product/2693">https://www.adafruit.com/product/2693</a></td>
</tr>
<tr>
<td>ADC</td>
<td>Convertor analogic-digital (ADC MCP 3008) with 8 channels. It allows to</td>
<td>1</td>
<td><a href="https://www.adafruit.com/product/856">https://www.adafruit.com/product/856</a></td>
</tr>
<tr>
<td></td>
<td>integrate the analogic sensors into the development board on the digital</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pins.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNS-MQ135</td>
<td>Analog Sensor for the gases (CO2, NH4, Ethanol) detection.</td>
<td>1</td>
<td><a href="https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/">https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/</a></td>
</tr>
<tr>
<td>Adafruit BMP 280</td>
<td>Digital Sensor for the temperature, pressure and altitude measurements.</td>
<td>1</td>
<td><a href="https://www.adafruit.com/product/2651">https://www.adafruit.com/product/2651</a></td>
</tr>
<tr>
<td>LEDs</td>
<td>Simple colored LEDs (Light Emitting diodes) for flagging the station status</td>
<td>3</td>
<td><a href="https://www.adafruit.com/product/1938">https://www.adafruit.com/product/1938</a></td>
</tr>
<tr>
<td></td>
<td>(online/offline/recording events)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection wires</td>
<td>Thin and small connections wires female-male (10cm).</td>
<td>20</td>
<td><a href="https://www.adafruit.com/product/3633">https://www.adafruit.com/product/3633</a></td>
</tr>
<tr>
<td>female-male</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini breadboard</td>
<td>Breadboard (46mm x 35mm)</td>
<td>1</td>
<td><a href="https://www.adafruit.com/product/65">https://www.adafruit.com/product/65</a></td>
</tr>
</tbody>
</table>

The development board (for the proof of concept station IoT Gateway) Raspberry Pi 3 Model B as an alternative to Nitrogen iMX board is a Single Board Computer (SBC) of a credit card size with a BCM2837 Quad Core (4x ARM Cortex-A53, 1.2GHz), 1GB RAM, Wi-Fi, Bluetooth 4.1 and Bluetooth Low Energy (BLE). Raspberry Pi has no internal flash memory and it uses a micro SD card for storage. The model also features 4 USB ports, HDMI port and 40 GPIO pins. The Raspberry Pi plate is powered by a Micro USB port that supports a 5V input voltage. Raspberry Pi 3B contains a GPIO 40 pins layout - digital inputs that can connect a lot of sensors / modules. There are two 5V pins, two 3.3V pins, and an 8 pin ground.
Raspbian is the official operating system released by Pi Foundation for the first time in June 2012. It is based on Debian and is very well optimized for Raspberry Pi ARM processors. The authors have used several distributions (Jessie and Stretch) and Java 8 SE-e (Standard Edition embedded edition), Python 2.7 and 3+ and node.js 8+ with Node-RED packages for the sensors data gathering.

All 40 GPIO pins on the Raspberry Pi card support digital inputs, so an analog-to-digital converter is needed to connect the analog sensors and read the values they provide.

The MCP 3008 is an analog-to-digital converter with which an analogue size can be accepted at the input and outputs a number representing an approximation of the analog signal of the input signal. This model works with 8 analogue channels, allowing up to eight analog sensors to be connected. The working voltage is 3.3V and the operating temperature is between -40 °C and 85 °C.

The SNS-MQ135 is an electrochemical sensor that measures air quality and is capable of detecting levels of carbon dioxide (CO2), ammonium (NH4), ethanol (C2H6O) and toluene (C7H8). The sensor has an analogue and digital output, uses a working voltage of 5V and has a current consumption of ~ 40mA. The working temperature is between -20 °C and 50 °C.

MQ9 is a gas sensor that measures the level of carbon monoxide (CO) and methane (CH4). Like SNS-MQ135, this sensor has analogue and digital output, a working voltage of 5V and a current consumption of ~ 40mA. The working temperature is between -20 °C and 50 °C.

The two sensors in the MQ series (MQ135, MQ9) require a pre-heating period of at least 24 hours, when commissioned for calibration.

SYH-2R is an analog sensor for detecting the relative humidity level. The working voltage is 5V and the operating temperature is between -20 °C and 85 °C.

BMP280 is a sensor designed by Bosch to detect temperature (± 1 °C accuracy) and barometric pressure (± 1hPa accuracy). It contains a voltage regulator and can use a working voltage of 3.3V to 5V. The operating temperature is between -40 °C and 85 °C. This sensor can communicate through both the SPI interface and I²C.

3.3. Hardware Diagrams

Fritzing is an open-source software product used to ease the connection of electronic devices and circuits. It allows the design of circuit diagrams and diagrams, as well as the design of PCBs. The schema and connection diagram were designed by Fritzing. When assembling the components, the mother-to-male connection wires and the breadboard were used to extend the 5V and ground lines (Figure 3 - Hardware diagram).
The analogue sensors are connected to the MCP3008 converter as follows: The SNS-MQ135 is connected to the analogue channel A0, the MQ9 is connected to the analogue channel A1 and the humidity sensor, SYH-2R, is connected to the A2 channel. The converter connects through the SPI interface to the Raspberry Pi board (Figure 4 - Schematic MCP3008).

The connection of the analog-digital converter to Raspberry Pi is done via the SPI interface (Figure 3 - Schematic MCP3008) as follows: VDD pin connects to 3.3V - GPIO 01, the GND pin connects to the ground pin - GPIO 06, the SCLK pin connects at SPI_CLK - GPIO 11, the MISO pin connects to SPI_MISO - GPIO 09, the MOSI pin connects to SPI_MOSI - GPIO 10 and the CE pin connects to SPI_CE0 - GPIO 08.

Interfacing the BMP280 temperature and pressure sensor with Raspberry Pi is done by the I2C protocol as follows: the SDI pin on the sensor connects to SDA1 - GPIO 02, the CS pin connects to SCL1 - GPIO 03, the GND pin to the ground-board on the breadboard and the pin 3.3V to the power plug on the breadboard.

Connecting the status LEDs is done through the following pins: Green LED connects to GPIO 21, blue LED connects to GPIO 12 and red LED connects to GPIO 05. All LEDs connect to line ground on breadboard.

3.4. Sensors Communications Protocols and reading data from the sensors

By using a communication protocol two or more entities in a communication system exchange information. The interface of the analog sensors (SNS-MQ135, MQ9, SYH-2R) and digital (BMP280) with the IoT Gateway development boards can be achieved through the following communication protocols:

- **Wired Sensors:**
  - **SPI (Serial Peripheral Interface)** - Serial Peripheral Interface is a synchronous communication protocol used to transmit information between master devices (such as IoT Gateway/Node development board) and peripheral devices. The SPI bus can operate with a single master device and one or more slave devices. In the authors solution, the analog-digital converter MCP3008 is connected to the IoT Nodes/Gateways development board by SPI, so the latter has the role of master and the ADC is slave. SPI uses four logical signals to communicate with the peripheral device. These are: serial clock (SCLK), master input slave output (MISO), master output slave input (MOSI) and slave select (SS). The clock is
configured by the master at a supported frequency less than or equal to the maximum frequency supported by the slave then the master configures the slave on the "low" level for the selected chip. At each clock cycle the data is transmitted in full duplex mode as follows:

- The Master sends 1 bit to Master-Output/Slave-Input and the slave reads it.
- The slave sends 1 bit to Master-Input/Slave-Output and the master reads it.

**I2C (Inter-Integrated Circuit)** – The Inter-Integrated Circuit (I2C) communication protocol was developed by Philips Semiconductor in 1980. It allows the communication of several slaves with one or more master devices. Similar to the SPI interface, Inter-Integrated Circuit is designed for small data transfer, but unlike SPI, the I2C protocol only needs two signal wires to perform data interchange. Each I2C bus is made up of two signal lines: serial clock line (SCL) and serial data line (SDA). The clock signal is generated by the master but some slave devices can force the clock to a low level to delay the master sending more data (or require more time to prepare the data before the master takes over).

**Wireless Sensors** – these sensors will be added in the next version of the solution:

- **BLE** – Bluetooth Low Energy will be used for direct connection with sensors which are linked to IoT Nodes instead directly in the IoT Gateway.
- **Z*-Protocols** – these will be used to decrease the number of IoT nodes and gateways and to increase the number of the smart sensors and also to optimize the power consumption on the edge devices (ZigBee and Z-Wave are more efficient in terms of power consumption that Wi-Fi):
  - ZigBee – now the published version is Zigbee 3.0, the complete IoT solution communications, from the universal language that allows smart objects to work together to the mesh networks. (Z-Wave protocol is used mainly in home automation and it is an alternative for ZigBee).

Data collection by the attached sensors is achieved both through the SPI interface (for communicating with analog sensors) and via the I2C interface (for communicating with digital sensors). To read data from sensors via I2C, the authors are using the Java DIO library (https://openjdk.java.net/projects/dio/). Bosch, the manufacturer of the BMP280 sensor, offers a Java implementation to retrieve the data. To read data from sensors via SPI we needed a way to interact with the MCP3008 analogue-to-digital converter. Therefore, the authors created an AnalogDigitalConverter class (Figure 5) that implements the Singleton design pattern and uses the Java jdk.dio (Device I/O) library (used in both Java SE-e Standard Edition embedded and Java ME-MicroEdition) to communicate with the GPIO pins. Next, authors created an AnalogSensor abstract class that implements the AnalogSensorOperation interface and contains an AnalogDigitalConverter, the analogue channel where the data is read and the gross value returned by the sensor.
public class AnalogDigitalConverter {
    private static AnalogDigitalConverter adc;
    private MCP3008 mcp3008;
    private MCP3008Config adcConfig;

    private AnalogDigitalConverter(int deviceNumber) {
        try {
            adcConfig = new MCP3008Config(deviceNumber);
            mcp3008 = (MCP3008) DeviceManager.open(adcConfig);
        } catch (IOException ice) {
            ice.printStackTrace();
        }
    }

    public static AnalogDigitalConverter getInstance(int deviceNumber) {
        synchronized (AnalogDigitalConverter.class) {
            if (adc == null) {
                adc = new AnalogDigitalConverter(deviceNumber);
                return adc;
            } else {
                return adc;
            }
        }
    }

    public void close() throws IOException {
        adc.mcp3008.close();
    }

    public int readChannel(int channel) {
        return adc.mcp3008.readChannel(channel);
    }

    public boolean isOpen() {
        return adc.mcp3008.isOpen();
    }
}

Figure 5. Partial implementation of the class AnalogDigitalConverter

Thus, all analog sensors will inherit the AnalogSensor class and implement the interface method to provide processing of the readable gross value.

3.5. IoT Cloud and Back-end Services

The application from the IoT Gateway communicates recorded data using the MQTT protocol with its implementation from Eclipse Foundation - Mosquitto MQTT. This implies the existence of a broker that has the role of selecting and filtering incoming messages. IoTP4mSCp uses cloud services such as Amazon EC2 IaaS – Infrastructure as a Service or Google Containers as a Service cloud to host the MQTT broker and the polluting station management / monitoring web application. If the authors use Amazon Web Services (AWS)/Google/Oracle IoT Cloud(s) PaaS – Platform as a Service, then the application will directly access the features provided by these IoT dedicated clouds.

3.5.1 IoT IaaS Cloud for data storage and processing

Amazon Elastic Compute Cloud (EC2) is a Infrastructure as a Service (IaaS) Cloud that offers a secure computing capability in the AWS cloud. EC2 courts can be resized and their number can be scaled according to requirements. These can be launched in multiple locations and geographic regions called Availability Zones (AZ). Each region comprises several AZs in distinct locations, interconnected by low latency networks.

Amazon EC2 supports multiple operating systems for which the users should pay the licensing fees such as: Red Hat Enterprise, SUSE Enterprise, Oracle Enterprise Linux, UNIX, Windows Server and free operating systems such as: Amazon Linux, Ubuntu, CentOS.

In terms of security, the cloud allows users to create security groups that behave like a virtual firewall that controls inbound and outbound traffic. Thus, specific ports for specific IP addresses can be restricted.

Internal storage of an instance data is done using an AWS – Amazon Web Services Elastic Block Store (EBS) instance. It provides data storage volumes and is automatically replicated for each Availability Zones (AZ) to provide high availability.
To create the EC2 instance, an Amazon Machine Image (AMI) must be selected that contains the software configuration (operating system, applications, preinstalled utilities). We chose Amazon Linux AMI 2018.03.0 (Figure 6) which is a RHEL 5.x-based distribution and includes AWS command line tools, Java 1.8, Python, Ruby, Perl. For the instance type we chose to t2.micro with 1vCPU of 2.5GHz and 1GB of RAM. The storage is provided by an 8 GB Elastic Block Store volume.

Figure 6. Amazon Machine Image (AMI) selection

For the full launch of EC2 we need to define a security group. One instance can be associated to one or more security groups. The security group used (called RDS-Rpi - Figure 7) for this instance is configured to allow inbound ports 22 - for SSH – Secure Shell, 8080 - to access the Web server of the solution and 1883 - for the Mosquitto MQTT broker.

Figure 7. Security group selection

After launching the EC2 instance (Figure 8), a private PEM key "aws-pi.pem" (PEM - Privacy Enhanced Mail file is a X509 digital certificate encoded in Base64) is downloaded to access the instance using the SSH protocol. Authentication is done with the user name "ec2-user".

Figure 8. Launching the instance from cloud Amazon EC2

The figure 8 shows the public IP of the back-end instance and the public DNS –Domain Name Service if it is necessary to be accessed from the Internet.

3.5.2 Data persistence and Server Side NoSQL database configuration

The MongoDB database is hosted by the mLab platform through Amazon Web Services. mLab offers a sandbox with a 500MB MongoDB instance. Adding a database user is done from the mLab platform through the Add database user form:

Figure 9. Adding a new user into MongoDB NoSQL Database via Amazon Web Services (AWS) mLab

Storage of collected data from IoT4mSCp IoT gateway is performed using a non-only SQL (NoSQL) MongoDB cloud database. Using a NoSQL database under this project is ideal for storing samples recorded by stations and processing them distributed through MapReduce.
MongoDB is a non-relational, open-source database developed in C++ and released publicly in 2009. MongoDB is document oriented, delivering high performance and scalability. Unlike a relational database where data is stored in tables, in MongoDB data is stored as JSON documents in collections. These collections do not impose a certain scheme, so the documents have a dynamic scheme and can have a different structure.

The persistence of the data recorded by the stations in the MongoDB database is carried out through two collections: stations - in which data is stored on the stations (their name, unique identifier, geographical position) - Figure 10.1 and probes - a collection in which the data persist recorded by a IoT Gateway development board station (air quality level, detected Wi-Fi networks and their signal) - Figure 10.2. Also, IoTP4mSCp solution has web interfaces which implies the existence of users who administer the stations. These are stored in the users’ collection in JSON format as in figure 10.3.

![Figure 10.1 The sample rate JSON structure](image1)

![Figure 10.2 The JSON for the station structure](image2)

![Figure 10.3 The JSON for the user structure](image3)

The JSON stored into the MQTT broker and then transferred to the MongoDB are next inserted into the distributed relational database for Big Data processing and reports dashboard creation.

The Server Side MQTT Mosquitto Broker configuration involves installation of the broker on the EC2 instance is done using the yum package manager. Restrict all non-specified subscription and publishing requests by setting the ‘allow_anonymous’ to ‘false’ value.

3.5.3 Node-RED framework

Node-RED is an IBM-based, flow-based programming tool for connecting hardware, APIs and online services. It is built on Node.js and uses its non-blocking and event-driven model, making it ideal for running on limited-cloud or cloud-based devices.

It provides a browser feed/flows editor that runs as default on the 1880 port, which can connect a lot of input, output, storage, and JavaScript functions. The feeds/flows can be imported / exported in JSON format – JavaScript Syntax Object Notation (e.g. {“software”: “Node-RED”, “version”: 1}).

Node-RED is a flow-based development tool developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.

Using a Node-RED instance (this can be deployed anywhere, even on localhost) and a development flow, the messages published via MQTT by the stations can be written to MongoDB instance.
This flow contains the following nodes (Figure 11):

1. MQTT output node (connected to mosquitto broker from AWS EC2)
2. JavaScript function node (collects and validates the payload from the MQTT message)
3. MongoDB input node (connected to mLab MongoDB instance)
4. Debug node (logs the raw MQTT message)

Figure 11. Flow in Node-RED on Server Cloud

3.6. IoT Communications protocols to the IoT cloud

As IoT middleware and communications protocols between the IoT gateways and the IoT Cloud for the solution: MQTT(s) and HTTP(s) – REST-API are used.

MQTT (Message Queuing Telemetry Transport) is a publish-subscribe-based communication protocol that runs over the TCP/IP protocols stack (Figure 12). Filtering messages is done on a topic-based basis by the broker, and messages are categorized on topics prior to publication.

Figure 12. MQTT within TCP/IP Communication protocols stack

In the above TCP/IP Communication protocols stack (Figure 12), the MQTT is an Application layer protocol and it does not provide security by design or by default. Indeed, MQTT has QoS-Quality of Service mechanism but this doesn’t replace the lack of the security.

Developed in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom), it was designed to be an extremely simple and easy protocol designed for devices with limited resources, low bandwidth, high latency, or unstable internet connection. The previous principles prove that this protocol is ideal for the machine-to-machine (M2M) and Internet of Things (IoT) domains where a small footprint is needed. The MQTT protocol has ports 1883 and 8883 (MQTT over SSL) reserved by IANA.
Since March 2013, MQTT has been standardized at OASIS - Organization for the Advancement of Structured Information Standards and after the protocol specification has been published [16], companies like Eurotech (formerly known as Arcom) have implemented the protocol in their products. Since 2016, MQTT has become an ISO standard (ISO / IEC 20922).

The publish-subscribe paradigm is a messaging model in which publishers publish messages that can be received by a subset or the entire subscribers set. The mechanism by which subscribers receive only a subset of all published messages is called message filtering. There are two forms of filtering: topic-based filtering and content-based filtering:

- In the topic-based approach, messages are categorized and published to topics (also called logical channels). Subscribers to the system will receive messages that are posted to the topics they are subscribed to. The sender is responsible for defining the message classes subscribers can subscribe to.
- In the content-based approach messages are delivered to subscribers provided that the attributes or content of the message correspond to the constraints defined by them. In this case, the subscriber is responsible for selecting and classifying the messages he receives.

Publishing subscriptions are managed by a broker which carries out the message filtering. All customers need to establish a connection with the broker. The client who publishes a broker message is called a publisher. The broker filters messages and distributes them to interested customers to receive a specific type of message. Customers who subscribe to the broker to receive messages are called subscribers.

An advantage of the publish-subscribe model is the loose coupling of subscribers. Publishers do not need to know of the existence of subscribers. Their main purpose is to publish a message to a particular topic.

After the broker has been configured, the MQTT client on the IoT Edge devices may run the MQTT client applications. When starting the application, an MQTT client is initialized to subscribe to the broker hosted on the Amazon EC2 instance at the following topics: /stations and /stations/{MAC_ADDRESS}, where MAC_ADDRESS is the MAC address of the network card of the IoT Gateway development board (e.g. for proof of concepts Raspberry Pi or Nitrogen iMX6). Subscribing to these topics is done to change the remote station configuration. All stations available on the platform can be modified by publishing a configuration to the topic /stations or only a specific station can be modified by publishing a configuration on a /stations/subtopic defined by the MAC address of the board.

For the implementation of the message processing the authors have defined a MQTTClien class that contains a Mqtt Paho client – by Eclipse - and implements the MqttCallback interface. This interface defines the messageArrived() method - Figure 13, in which the current configuration of the station is updated.

```
@override
public void messageArrived(String topic, MqttMessage message) throws Exception {
    if (MQTTConstants.MQTT_CONFIG_LOCAL.equals(topic)) {
        MQTTConstants.MQTT_CONFIG_ALL.equals(topic)) {
            final PocFacade pocFacade = PocFacade.getInstance();
            Gson gson = new Gson();
            try{
                StationConfig stationConfig = gson.fromJson
            (new String(message.getPayload(), "UTF-8"), StationConfig.class);
                pocFacade.propose(stationConfig);
                new Thread(pocFacade).start();
            }
        }
    }
}
```

**Figure 13.** MQTT messageArrived(...) method implementation
In the above method (Figure 13) the messageArrived() method is called by the Paho MQTT library within the JVM – Java Virtual Machine with two parameters: the topic name and the message. The received message is encapsulated within GSON object and then the station configuration is modified by a new thread within JVM. Any occurrence of an exception is captured by the Java try-catch mechanism.

3.7. RESTful Services implementation

RESTful services are implemented through the classes: ProbeController, StationsController, UsersController. They are annotated with @RestController and @RequestMapping specifying the URI of the resource. Adding a polluting sample (Figure 14) is done through a POST HTTP request to "/api/probes".

```java
@PostMapping(value ="", "/")
public ResponseEntity save(@RequestBody Probe probe){
    if(probe == null){
        return new ResponseEntity(HttpStatus.BAD_REQUEST);
    }
    Probe savedProbe = null;
    try{
        savedProbe = probeService.save(probe);
    }catch (Exception ex){
        LOG.error(ex.getMessage());
    }
    if(savedProbe!=null){
        return new ResponseEntity(savedProbe, HttpStatus.CREATED);
    }else{
        return new ResponseEntity(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Figure 14. POST Probe endpoint

The POST HTTP request must contain a request-body in which the entity is defined in JSON format. If the request does not contain a valid body, the client receives an HTTP 400 - Bad Request error.

3.8. Front-end implementation

The IoT4mSCP solution's Frontend component is a Single-Page Application (SPA) application built into the Angular 5.2 based framework and the Angular Material library - used to use predefined components such as dialog windows, buttons, and tables. The project was created using the Command Line Interface (CLI), which facilitates the generation of components, services and modules.

The user interface includes the following sections: user area - includes authentication, account creation and profile modification, station area - fetches information about IoT Gateway stations, adding / configuring / deleting them and the pollution sample area where detailed reports are presented based on recorded samples Figure 15).
In figure 15 for each device a list of devices from the IoT Solution is shown. Adding and configuring devices is straightforward. The user experience is enhanced with the dashboards such in figures 16 and 17.

The Stations component is responsible for interacting with the Raspberry Pi pollution monitoring stations. It contains the following sub-components: StationsList - a component that defines a list of stations that is displayed in a grid (Figure 16), StationsCreate - component that defines a dialog with a form to create a new station, StationsMap - component showing the geographic position of each station on the map, StationsConfig - a component that displays a dialog with the station configuration options.

Adding a station is done through the StationsCreate component (Figure 17) that includes a modal dialog with a stepper component in the Angular Material library. Adding a station involves three steps: filling in the station details (name, identifier), filling the station position (latitude, longitude) and confirming the entered data. Upon adding a station, a POST HTTP request is made to /api/stations to the solution backend.
For the data presentation and analysis, we used the ng2-charts guidelines that are based on the Chart.js library. The data-analysis component injects the probes, service through which REST services are called from the P.M.A backend component to retrieve the pollution samples. These are retrieved according to the user defined parameters: the station for which the display is to be displayed, the parameter list for the analysis, the start date and the end date.

Figure 18 shows the real time monitoring of the indicators reported from the IoT gateways distributed within the city:

Since the solution’s frontend component is a single-page application, it is necessary to map the requests from Springboot to Angular. For example: When receiving an HTTP GET request of the resource identified by localhost: 8080 / dashboard, the Springboot framework looks for a class controller that can handle that request. Therefore, interpreting applications by the P.M.A frontend component is done by redirecting them using the following ViewController class (Figure 19):
In figure 19, the controller class is responsible for creating objects which are handling the flow of the HTTP requests from the web dashboards, in order to access different REST resources or even HTML webpages.

The first prototype implementation did not take into account the security – at least the communications authentication and data confidentiality – encryption. The next section presents the MQTT secure option, because by design MQTT has no security.

4. IoT Security Challenges and the solution merits

The PoC and the proposed solution of the authors in this paper had several security challenges and areas of enhancements and merits. The first subsection of this section highlights the security challenges while the second subsection pinpoints the merits and extensions of the approach.

4.1. MQTT communication protocol and the security challenges

MQTT does not contain security features built-in to the protocol. The reason is to keep it simple and lightweight. Another reason is related to IoT hardware. Usually, the IoT devices have limited computing power and memory capacity [15]. The hardware resources are insufficient for the support of complex cryptographic. Another IoT security challenge refers to the reliability level of the networks used to maintain the IoT devices.

A security-related implementation exists within version 3.1, where the credentials could be sent through the computer network in a MQTT packet. Encryption of the MQTT packets is handled by SSL/TLS protocol or applications providing encryption of the data. The reserved TCP/IP port for MQTT by using SSL/TLS communication is 8883.

The MQTT documentation states several security challenges [16] that must be handled by the solution providers. Those solution implementations must include mechanisms for authentication, authorization, integrity and privacy. Guidelines to deploy a MQTT solution consistent with NIST Framework for Improving Critical Infrastructure Cybersecurity are provided by [17]. The NIST Cybersecurity Framework [18] provides specifications and mechanisms for organizations to identify the possible improvements of the cybersecurity risk management. In the context of MQTT, this framework is called MQTT Cybersecurity Framework. The core functions of the MQTT Cybersecurity Framework are used to make assessments of the cybersecurity level for the organization that uses a deployed MQTT solution. The core functions target the following aspects [17]:

1. Identification – establishes what are the systems, assets, data, and capabilities that need to be protected in the context of the MQTT solution exploitation. Also, prioritization and processes to achieve the risk management goals must be established.
2. Protection – develops and implements the right measures to protect or prevent undesirable actions over the MQTT related items identified by the previous function.
3. Detection – develops and implements the right measures for MQTT cybersecurity event occurrences.
4. Response – establishes the appropriate counter measures when a MQTT cybersecurity event occurred.
5. Recover – aims the activities to restore the organization infrastructure when a MQTT cybersecurity event caused significant damages.

The security levels supported by a MQTT solution are:
- Network – a trusted connection could be provided by physically secure network or VPN usage for all communication within the MQTT infrastructure.
- Transport – encryption ensured by SSL/TLS protocols. Identity of the both sides of the communication is proofed by the secure transport protocol implementation.
- Application – authentication of the devices is ensured by username/password credentials provided by the MQTT protocol. Also, encryption of the payload is possible, but constrained by the hardware specifications of the IoT clients.

Even though SSL/TLS usage introduces overhead in network communication and asks for more CPU on client side, the transport level encryption is critical to protect the communication within the MQTT infrastructure. When the transport level encryption is not feasible due to constrained devices the alternative approach is to least encrypt the payload of several packet types.

Beside the payload encryption, a stamp attached to the packet PUBLISH could be used. The format of the stamp could be: digital signature, message authentication code, checksum, or hash algorithm.

A different aspect of the SSL/TLS usage is to validate X.509 certificate on client side to prevent several types of attacks over the secure connection. The X.509 certificate is provided by the MQTT broker. The recommendation is to use X.509 certificates from trusted CAs.

By using X.509 client certificate, security improvements are added to the MQTT infrastructure [19], such as:
- Identity of the MQTT clients is validated.
- MQTT clients are authenticated at transport level.
- Some measures as responses to the MQTT cybersecurity event are taking place (e.g. lock client before the packet CONNECT is sent).

In addition to SSL/TLS usages, the following security mechanism could be used: payload encryption and signature, and authorization mechanisms to limit the client access to the MQTT infrastructure resources.

In MQTT, the authentication with username/password is provided by the Control Packet called CONNECT. The presence of the connect credentials within the Control Packet CONNECT is signaled by the Connect Flags section of the packet, figure 20 [16].

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User Name Flag</td>
<td>Password Flag</td>
<td>Will Retain</td>
<td>Will QoS</td>
<td>Will Flag</td>
<td>Clean Session</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>byte 8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 20. Connect Flags Section

In figure 20, for both credentials flags, the value 0 means not present. Password flag is depending on the User Name flag being 0 mandatory if the User Name flag is set to 0.

The MQTT username/password credentials are stored in plain within the payload if the flags are set in the header side of the MQTT packet. If the User Name is set to 1, then an UTF-8 encoded string is stored within the payload. If the Password flag is set to 1, then a field containing up to 65535 bytes is stored as password within the payload section. As result of processing the packet CONNECT with username/password credentials, the server is able to do authentication and authorization of the device to access resources within the MQTT deployed solution.

In addition to username/password credentials, MQTT clients could provide different items to server in order to do the authentication like client identifier or X.509 certificate.
By other hand, the username/password credentials could be used to implement OAuth 2.0. That open protocol is used to allow a secured authorization of the client to the MQTT infrastructure resources based on tokens. A MQTT client is able to connect to a certain broker or to publish or subscribe with an access token.

Beside the MQTT protocol specification, additional guidance related to secure MQTT solution implementation could be found on OWASP platform [20].

4.2. Merits of the proposed solution

The solution approach has several area of merits in terms of scalability, reliability, security and HA – High Availability. The merits of the approach presented in this paper are the following:

- Easy integration of the vehicles traffic management system (e.g. traffic lights) with the authors proposed solution API as pluggable component. These API are easy to be exposed to any traffic management system in order to create dynamically different routes in order to avoid vehicles congestion.

- The design of the architecture is suitable for scalable extension and HA – High Availability systems implementation. The architecture and infrastructure used for the pollution indicators collection within an IoT smart city solution presents the scalability because is combining asynchronous message processing (MoM – Messages oriented Middleware / MQTT brokers) with IoT Clouds and NoSQL databases for fast data insertions and potential extensions with Big Data processing form the relational databases.

- The deployed IoT gateway software is developed mainly in cross platform software development technologies - e.g. mainly in Java and some optional modules with Node-RED/node.js - and the software is portable and compliant with different development boards/MCUs - e.g. iMX Nitrogen, Raspberry Pi, etc. - and industrial IoT Gateways - Eurotech, HMS Netbiter, Dell, etc.. The design of the IoT Architecture for collecting pollution indicators within a smart city is using the best practices in software development field.

- Security of the MQTT communication protocol implementation for the collected data represents a valuable feature, because if the data is altered by a malicious attackers vectors from the IoT edge devices to the IoT cloud backend services, then the real-time decisions of the traffic management can be dramatically affected and with impact in massive congestions.

The proposed solution can be enhanced in several areas from the data interpretation and systems integrations to cybersecurity. Some of the afore mentioned ideas are detailed as future work in the next section.

5. Conclusions, data interpretation and future work

After the data is collected from the sensors, the data interpretation process begins. For the moment, the data interpretation is subject of improvement and needs to be enhances with multiple features. As an example, we have a few values in figure 21 showing the pollution sample rates stored from April 2018, and in figure 22 from December 2018, in a major city – Bucharest, Romania, European Union.

Figure 21 and 22 show partial data collected from IoT gateways deployed within the city and emphasizes according with the table head the following information: the station identification MAC address and the deployment location; the collected data – temperature, humidity, pressure, altitude, CO2, NH4, CH4, CO, toluene, Wi-Fi and noise decibels; and the date – time of the collected data.
The authors have the full database with the pollution values from different places of Bucharest in year 2018 and they will consider WHO database format for reporting directly to World Health Organization. But meanwhile, in the same area and in the same indicators (comparing the carbon-monoxide and carbon dioxide from the same stations in April and in December 2018), can be observed how the pollution in December 2018 is higher than in April 2018, because of the following:

- Seasonal influence on the cold weather outside and shopping periods for the Christmas traffic, but also thermo-central consumption of gas for heating the population households. Also, in 2018, the transit pipes for hot water from the city were more damaged than in 2016 and this can not be done without investments.

- Regulations and laws changes – in 2018, it was much easier to bring very old cars - taxes were lower - that produce higher pollution than in 2016-2017.

- The number of the powerful Wi-Fi routers increased in 2018 compared with 2017 and also the decibels produced as noise pollution by Wi-Fi access points.

- Price of the gas within gas distributors chains/stations influenced the traffic.

- Other factors that may influence the pollution but for which a direct relation cannot be confirmed – covariance between the factor and the pollution level.

Figure 21 shows for instance, various areas from Bucharest with the pollution indicators (including the decibels generated by the noise and Wi-Fi networks) in analytical comparison between April 23rd, 2018 and December 6th, 2018. The authors will enhance the solution in several areas. Data interpretation and analytics is one of the first focuses as it proves the correlation between different factors. The current interpretation is based on the statistical and business intelligence algorithms, without involving predictive analytics such as SAX – Symbolic Aggregate approXimation [28], [29] or more elaborate AI – Artificial Intelligence / data-mining algorithms. Also, Big Data technologies as in [31] and Apache Spark platform may be used as alternative to the SAX in order to produce consistent interpretation regarding the collected data.

The pollution information has been summarized in various diagrams/dashboards/maps such as the one from figure 23:
The second area of improvement is the IoT edge devices replacement with IP67 outdoor resistant devices for all three layers: sensors, IoT nodes and IoT gateways. Additionally, the ratio between the number of gateways and sensors needs to be adjusted when more nodes with relatively low cost but good battery life and communications features will appear. The sensors types and calibrations can also be enhanced as for the PoC the authors did not use professional sensors from the market.

The cyber-security can be optimized. The authors focused on the security of communications between the IoT gateways and clouds but there is room for more. By using Java Card 3.1+ technology [30] within different form factors (iSE – integrated Secure Element, eSE – embedded Secure Element, etc.), the authors are targeting not only to ensure the security of the end to end communications (from sensors to cloud), but also to improve the node / gateway security, by running sensitive and computational cryptographic code on the temper resistant Secure Element with Java Card 3.1+.

The authors are also targeting to use calibrated high quality sensors and interface implementation for reporting data directly to World Health Organization, in order to maintain the quality of data for the future work.

As future work, the authors will target a case study for applying the PoC with enhancements in different medium cities in cooperation with the local authorities to see the traffic impact simulations and the integration with other smart city systems, because the existing projects such as AirVisual [32] are limited. The AirVisual project is limited only to monitoring the WHO PM2.5 (fine particulate matter) and it does not analyze further the harmful particles for the humans’ health. In figure 24 [32] the one may observe the top of polluted cities in Romania in 2018.
In figure 24, Bucharest is on place 4 of the most polluted cities from Romania (according with IQAir AirVisual platform), where the average PM2.5 is at value of 20.3 and the worst months in terms of pollution are January and December with PM2.5 values of 26.1 and respectively 27.7. As conclusions, the cars and the transportation traffic is only a part of the problem, because the industry development, house / apartments HVAC (Heat, ventilation, Air-Conditioning) solutions, waste management and the recycle mechanisms must be improved as well.

The authors focused on the Bucharest traffic congestion as, according with TomTom index [33] in year 2018, the city has been reported as the first place with the biggest value regarding the traffic congestion and in place 10/11 world wide after Mumbai, Bogota, Moscow, Istanbul and other major cities. The congestion level in year 2018, according with [33] was 48% with 1% improvement than year 2017 and with best day on Sunday, April 8-th and the worst day on Wednesday, December 5-th. The proposed approach of the authors is “pluggable” in terms of the integration with traffic management systems of the smart cities. It can dynamically determine, by making use of the drivers smart-phone applications and of the traffic light management, routes which diminish congestion and help in avoiding the polluted areas.

Author Contributions: C.T. – overall conceptualization; investigation, methodology, resources, software and writing - original draft for chapter 5; A.A. and M.P. – investigation; methodology, resources; software development (mainly A.A.), experiments and analysis of results; A.Z. – methodology, introduction and conclusions; writing – original draft preparation; writing – review and editing

Funding: This paper presents results obtained within the PN-III-P1-1.2-PCCDI-2017-0272 ATLAS project (“Hub inovativ pentru tehnologii avansate de securitate cibernetică / Innovative Hub for Advanced Cyber Security Technologies”), financed by UEFISCDI through the PN III –“Dezvoltarea sistemului national de cercetare-dezvoltare”, PN-III-P1-1.2-PCCDI-2017-1 program.

References

13. uRADMonitor Alba Iulia [Online]. uradmonitor.com/alba


32. World Most Polluted Cities in 2018 – AirVisual project: https://www.airvisual.com/world-most-polluted-cities?continent=59af92ac3e7000c1bd78e52&country=zk4LhMNzkk6ky46az&state=&page=1&perPage=50&cities=


© 2019 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).