ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ
V PRAZE

Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

2018 Vadim Kharchenko

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ
V PRAZE

Fakulta elektrotechnická

Katedra mikroelektroniky

INTELLIGENT INTERFACE FOR ENVIRONMENTAL MEASUREMENT

Inteligentní rozhraní pro měření okolního prostředí

únor 2018

Bakalant: Vadim Kharchenko

Vedoucí práce: Ing. Adam Bouřa, Ph.D.

Čestné prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci zpracoval sám s přispěním vedoucího práce a konzultanta a používal jsem pouze literaturu v práci uvedenou. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé bakalářské práce nebo její části se souhlasem katedry.

Datum: 09.01.2018

podpis bakalanta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kharchenko** Jméno: **Vadim** Osobní číslo: **392833**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Komunikace, multimédia a elektronika**

Studijní obor: **Síťové a informační technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Inteligentní rozhraní pro měření okolního prostředí

Název bakalářské práce anglicky:

Intelligent Interface for Environmental Measurement

Pokyny pro vypracování:

1. Seznamte se s problematikou využití jednodeskových počítačů pro účely zpracování senzorových dat a jejich prezentaci pomocí webové aplikace. Prostudujte principy komunikačních rozhraní, která se pro tento účel využívají.
2. Na základě získaných poznatků navrhněte inteligentní rozhraní pro připojení různých typů senzorů. Navrhněte systém tak, aby automaticky rozpoznal připojení nového senzoru a byl schopen zpracovávat jeho data.
3. Sestavte testovací pracoviště a ověřte funkčnost vašeho návrhu s alespoň čtyřmi typy senzorů. Navrhněte případná vylepšení.

Seznam doporučené literatury:

[1] Garlík, B.: Inteligentní budovy, BEN - technická literatura, Praha, 2012, ISBN 978-80-7300-440-8.
[2] Vedral, J., Fischer, J.: Elektronické obvody pro měřicí techniku, Vydavatelství ČVUT, Praha, 2004.
[3] Halfacree, G., Upton, E.: Raspberry Pi uživatelská příručka, COMPUTER PRESS, 2013, EAN: 978-80-2514-116-8.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Adam Bouřa, Ph.D., katedra mikroelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **04.10.2017** Termín odevzdání bakalářské práce: **09.01.2018**

Platnost zadání bakalářské práce: **28.02.2019**

Ing. Adam Bouřa, Ph.D.		prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání	Podpis studenta

Anotace:
Tato práce je věnována analýze senzorové sítě složené z Raspberry pi3, Arduino Nano a senzorů. První část se zabývá teorii senzorů a teorii související s podobným systémem. V druhé části práce jsou popsány použitý hardware a software, a také měřicí systém. V poslední části jsou demonstrovány výsledky měření a možný vývoj systému.
Klíčová slova:
Senzor, Raspberry, Arduino Nano.

Summary:
This work is devoted to the analysis of a sensor network composed of Raspberry pi3, Arduino Nano and sensors. The first part deals with the theory of sensors and theory related to a similar system. The second part describes the used hardware and software, as well as the measuring system. The last part demonstrates the measurement results and possible implementation improvement.
Key words:
Sensor, Temperature Sensor, Humidity Sensor, Dust Sensor, CO2 Sensor, Raspberry, Arduino Nano, Intelligent System.

Contents

Abstract

Usually, smart-something, like smart home, smart car, etc. differ from mundane examples of the same things by having sensor systems and/or remote controls installed and operational. For example, smart home generally means automation of lighting, heating/air condition and security, which in turn means collecting relevant data, such as - for heating/air conditioning - current temperature, humidity, current state of windows, air conditioner and other things that may have influence on values collected by aforementioned sensors, and then either automatically change some state, like turning on heater or a bathroom ventilation fan.

Although there is a little problem with smart homes, the thing called in programming and technology "Bleeding Edge", which, applied to smart home means that a lot can be done, but usually, people are perfectly fine without this technology even if it would be cheap, or free.

1 Tasks

1. Research the ways and issues of Single-Board Computer use for the purpose of sensor data processing and said data demonstration via web application. Research the principles used for in-system communication in such a set up.

2. Based on research in point 1, design an intelligent interface suitable for connection of multiple sensors of different types. Design it in a way, that a newly connected sensor would begin its functions and get its data evaluated, refined and saved without the need of system administration.

3. Assemble a sensor system and test its functionality with at least four different sensor data types. Propose appropriate ways for improvement.

2 General Theory

2.1 Sensors [1]

By definition - a sensor is an element, or a whole device, purpose of which is the detection of events or assessment of some specific electrical, physical, or chemical values in its environment and conversion of these into some other type of an easier measurable value, in the vast majority - to some electric value.

Sensors can be differentiated by five main attributes:

- By measurand, or the primary input quantity
  - Electrical - voltage, current, resistance, etc.
  - Physical - temperature, pressure, illumination level, etc.
  - Mechanical - shift, tilt, position, speed, etc.
  - Chemical - levels of specific gases or elements in given environment
  - etc.
- By the transduction principle, or physical principle of measure - resistive, capacitive, inductive, magnetic, mechanical, piezoelectric, semiconductive, optical, chemical, etc.
- By whether it is a touch or proximity sensor
- By energy supply requirement
  - Active - if the measurand affects the sensor in such a way, that it becomes a source of electric energy
  - Passive - if the measurand only changes the sensor's properties so that it has to be additionally evaluated, e.g. resistive sensors. This type of sensors has to be externally powered
- By construction principles - electrical, electromechanical, electrochemical, electromagnetic, pneumatic, semiconductive, optoelectronic, etc.

Every sensor has a number of attributes, such as

- Sensitivity - a ratio of magnitudes of the output signal to the input signal. Basically, it defines amount of change in output signal following a small change in input. The less sensitivity a sensor has - the harder it is to determine the right input value using the output information.
- Resolution - the smallest increment in measurand, that creates a difference in output. Equals to the smallest change in input that can possibly be seen in the output.
- Threshold - the minimum value of the input, under which the output equals zero. When the input value is above threshold, the sensor begins to give output.
- Range - the maximum input value under which the sensor is designed to be used, e.g. if a thermometer is intended to be used to measure a temperature of 100 degrees Celsius, it must have this point on its scale, or to have a minimum range of 100°.

- Span - the difference between the range and the threshold, so a thermometer scaled from -20°to +50°has a span of 70°.
- Precision - a degree of how close to each other would statistically be the sensor's readings of the same input. For example - if when measuring the same value, one sensor shows [5.1, 5.2, 5.1], it has better precision than the one showing [5.0, 5.5, 5.8].
- Accuracy - a degree of how close to the actual value are the sensor's readings. If we take the example from above and say that the real value those sensors measured is 5.5, the second one would be more accurate.
- Hysteresis - a degree of how much would the sensor's readings of the same input differ if approached from opposite sides. Low hysteresis means that the save input value should be evaluated the same, regardless of how it came to current state.
- Non-Linearity - description of a mathematical dependency of output on input, which usually is considered best to perfectly follow the y = k*x formula. Non-Linearity is the deviation between such direct dependency and real input-output relationship.
- Offset - non-zero output value over the zero input value.
- Environmental Specification - temperature, pressure, humidity and other parameters' ranges inside which the sensor is intended to use and can give reliable readings.

2.2 Microcontroller boards and Single-Board Computers

Single-board microcontroller is a board with the circuitry needed for performance of a control task: a microprocessor, I/O pins and circuits, clock generator, RAM, secondary memory, and integrated circuitry. The main purpose for these is the help with application development, so that no custom controller hardware is necessary.

Important part of SBMs is their inability to run an operation system. Instead, they repeatedly run only one program.

On the other hand, Single-Board Computer means a complete computer, built on a single circuit board. Technically, blade servers and most laptops count as such, although nowadays in most cases when the term is used, it means something smaller and not as powerful. Typically SBC differs from SBM with additional more powerful hardware, operation system and non need in external configuration - it can be done from the SBC itself.

3 Theory of a sensor system

Even the simplest smart home has to include a number of points to operate as such:

1. Sensors - basically, the automatic input makes the difference between a smart home and a remote control.
2. Information transfer - whatever data the sensors collect has to be transferred somewhere else for collection, processing, and visualization because sensors themselves are usually unable to do any of this.
3. Control device - basically a computer, it has to be able to be used for sensors administration, data collection, processing and visualization, and whatever activation of the remote control the system is implied to have.
4. Remote Control (optional) - any kind of servo motors, electric and electronic activation/deactivation of circuits or appliances, mostly for the system's ability to do it all automatically or for ease of access, such as using a smartphone as a remote.

3.1 Sensors

There is obviously a tremendous amount of different types of sensors for different needs, but the option relevant to this work is the way they communicate with whatever they are connected to:

- Analog - e.g. the simplest thermo-resistance sensors
- Discrete - mostly either some counter or analog input pre-converted in sensor
- Pulse - e.g. flow sensors, sending pulses every time a full wheel turn is achieved
- Digital - especially the so-called one-wire kind, sending all the information in an easily readable format, which is very useful in the case that the sensor collects more than one data type. Easier to set up and usually sensors like these have fewer output wires, which means the ability to connect more of them to one board.

3.2 Single-Board Microcontrollers and Computers

As has been said above, sensors are mostly made without any kind of built-in systems that would make it possible to convert whatever data they "sense" into a readable format and then send it wherever else, so every sensor has to be connected to a controller, like one of Arduino microcontroller boards, or straight to an RPi board, or somewhere else with its own processor and pins for connection.

While reading this work, the reader may get a feeling that Arduino and Raspberry Pi are the only types of microcontrollers on the market, which is of course not true. There are different types and manufacturer companies, e.g. Asus Tinker Board, BeagleBoard from Texas Instruments and a whole lot of Chinese brands that may or may not have been started as an effort to make a cheaper copy of RPi, Arduino or some other well-known brand.

Picking the right one is mostly dependent on the requirements for the system capacities, like RAM and CPU rate, initial and upkeep costs, and reliability. Aside from the capacities point, it would probably be always a problem to pick the perfect balance between price and reliability, especially with the latter one being so obscure even after actual testing.

The Raspberry Pi 3B - Arduino Nano conjunction is used in this work as an example of an assembly leaning towards the reliable end - the last and the strongest board of Raspberry Pi - one of the most well known brands in the field, which makes us believe that they would not risk their reputation with making low-quality product, and one of the smallest Arduino boards - just enough for our purposes, but with the same guarantees related to the brand name.

Figure 1: Raspberry Pi 3B on the left and Arduino Nano on the right

3.3 Data Transfer

Basically, it's about the way how the data from a sensor can be transferred to the destination - control device. The transfer types are not so numerous, at least the useful ones. First, it's mostly some shorter wires to e.g. an Arduino, or some other platform from which a sensor can be powered, activated and to where the data initially go, but then there is a not so numerous amount of ways to send data from Arduino to our control device, which include:

- Cable - mostly USB or twisted pair, with their pros and cons, such as high reliability at almost no cost, but over a very limited effective distance between points, especially if low-voltage devices like Arduino is used.
- Wi-Fi or Bluetooth - may be harder to set up, but ultimately easier to make it aesthetic, which is useful when it is intended to use in homes or offices instead of some places with few people around, like e.g. warehouses. There is also an option of simpler RF link modules without built-in security, but they should be used on one's own risk, only for non-vital purposes.
- GPRS - unlike those before, is not free of charge, and most probably the payments it needs must be regular - even if small, just to keep the sim card from expiry, but obviously this option works everywhere except for somewhere underground.

3.4 Control and Administration

There also should be an endpoint database to collect all the measured data - probably not exactly necessary, but generally easier to implement and then read the results. The first obvious option is to use a PC, but PCs are relatively loud and energy consuming, so it may be hard for people to make them work in 24/7 active mode. Next option is some variant of a server machine, but it generally ranges from loud and hungry for electricity to something not very powerful, and it always would have sizes starting from those of a regular laptop. So there is a good alternative to both of those: a single-board computer, like Raspberry Pi. It is absolutely quiet, for there are no coolers in basic equipment, it is small - mostly on a scale of a thick credit card, and it is specifically designed for low energy consumption. Of course, the cheaper single-board computers can't be as powerful as actual PCs or server machines, but unless one would want to make an industrial-scale setup, or to support hundreds to thousands of simultaneous user interactions - which is definitely not expected when making a home-based system - this power should be enough to accomplish the task assigned to it.

Usually, it's assumed that a single-board computer would be used in conjunction with external monitor and input devices, but it can be avoided by connecting to it from some other computer remotely via e.g. SSH or VNC protocols. After such connection is set up and secured, any changes may be uploaded from any place with Internet connection.

Or one can use an opposite approach for the sake of security: Internet is full of bots attempting to break their way into all the systems they can find by connecting to every possible IP on every possible port, especially the aforementioned SSH and VNC default ports, 22 and 5900 respectively, and find the user id - password pair with brute force. The easiest way to stop them from breaking in is to just leave the least number of ways in possible, namely by disabling any kind of connection except for viewer purposes from outside the local network. This option may seem very inconvenient, but to be fair - such systems should rarely need any tweaking after they've been set up.

3.5 Data Processing

The process of data transformation from raw sensor output to an easily accessible set of datapoints consists of these steps:

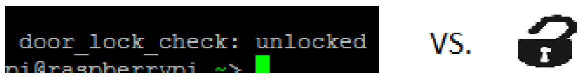
- Evaluation and transformation of sensor-specific signal set into a generally readable format by the programmed logic of whatever they are directly connected to - e.g. Arduino programmed to read the raw sensor data and encode it into JSON

- Aggregation of these already readable values - optional, but especially valuable in case of a big number of such inputs, for it can be used to e.g. combine a hundred datapoints that came inside one second, into one datapoint for this second,

- Placing the aggregated data into a storage database, from where it can be accessed later.

3.6 Visualization

Of course, text shell-based visualization may be enough for strictly technical use, e.g. when the system is set to automatically send a text message if air humidity in a warehouse has gone above a pre-set limit, but if the measured data is intended to be regularly seen by people, especially someone who's "far from all things technology", it would be much easier to understand and use graphical interface.



```
door_lock_check: unlocked
pi@raspberrypi ~>
```

Figure 2: An example of how the same message can look.

For the sake of visualization any system would need access to a database with the collected data, either communicating to some remote server, or having local software capable of processing and visualization on the control station itself.

3.7 Remote Control

Only collecting data for surveillance without any kind of control over the system's output may be enough for some specific simple uses, but even if some action is not exactly hard to perform physically, connecting some electric switch, motor, or some computerized system to control them remotely can save some time every time they have to be used, or after programming them to activate automatically on timer or when some specific conditions are met - it would save even more time. And obviously, in some conditions time actually does mean money, so such systems may very well not only bring comfort but even be profitable.

4 Practical Part

4.1 Build

The build used for this work is the following:

- Raspberry Pi 3B - 1.2 GHz CPU, 1 GB RAM
- Arduino Nano - 32 kB flash memory
- DHT11 temperature and humidity sensor
- Sharp GP2Y10 optical dust sensor
- MH-Z19 CO2 sensor
- Micro USB 5V 2.1A adapter as a power source for RPi
  - Optional - power bank as a power source for RPi at field measurements
- USB to Mini USB cable between RPi and Arduino
- Breadboard, jumper cables, resistor, capacitor

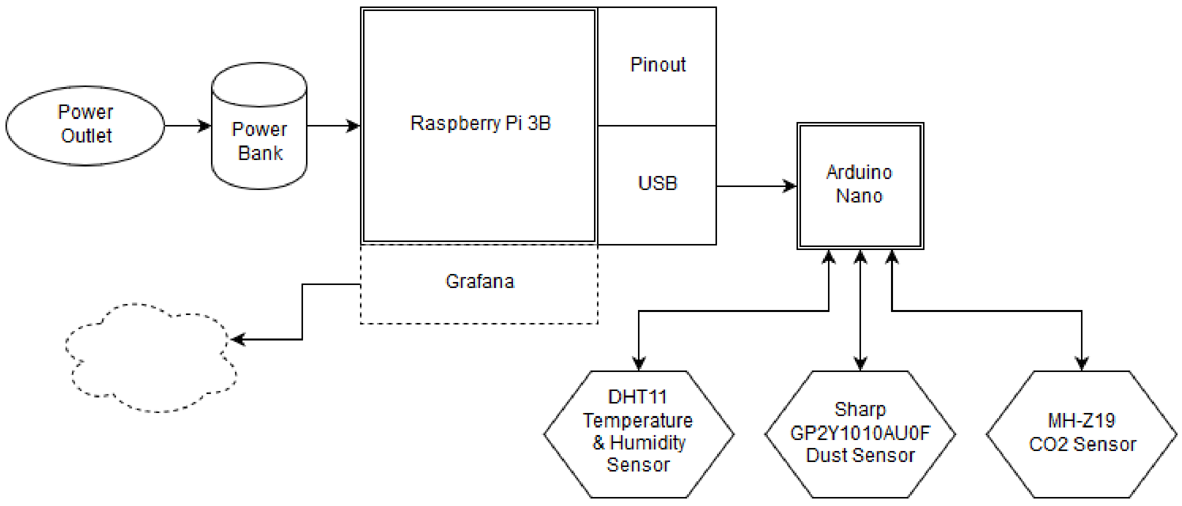


Figure 3: System connection scheme.

RPi is connected to either a micro USB adapter 5V 2.1A or a power bank through a USB to micro USB cable 5V 2.1A.

Arduino Nano is connected to RPi with a USB to mini USB cable, that is used for data transfer as well as a power feed, and to a breadboard with all its pins. For the purposes of programming Arduino board, RPi has Arduino IDE, so as to reduce the effort of starting the board's work to picking all the settings from menus and writing a code using a set of C/C++ commands for the board to follow. Although in this case Arduino programming was mostly done from the Arduino IDE on a PC because said PC works considerably faster than RPi when managed from the same PC.

All the sensors, following the schemes below, are connected to the breadboard and then to Arduino via jumper cables, and in case of the Sharp dust sensor - with an additional resistor and a capacitor.

The DHT11 sensor consists of a humidity sensing component, an NTC temperature sensor (or thermistor) and an integrated circuit on the back side of the sensor. The humidity sensing part consists of two electrodes with moisture holding substrate in between so that the substrate's conductivity changes with changing air humidity. The conductivity change is then measured, processed and translated by the integrated circuit. The thermal sensing element in this sensor is based on the principle that the polymer or ceramics it's made of, decrease their resistance with rising temperature. Again, the IC measures and processes the resistance.

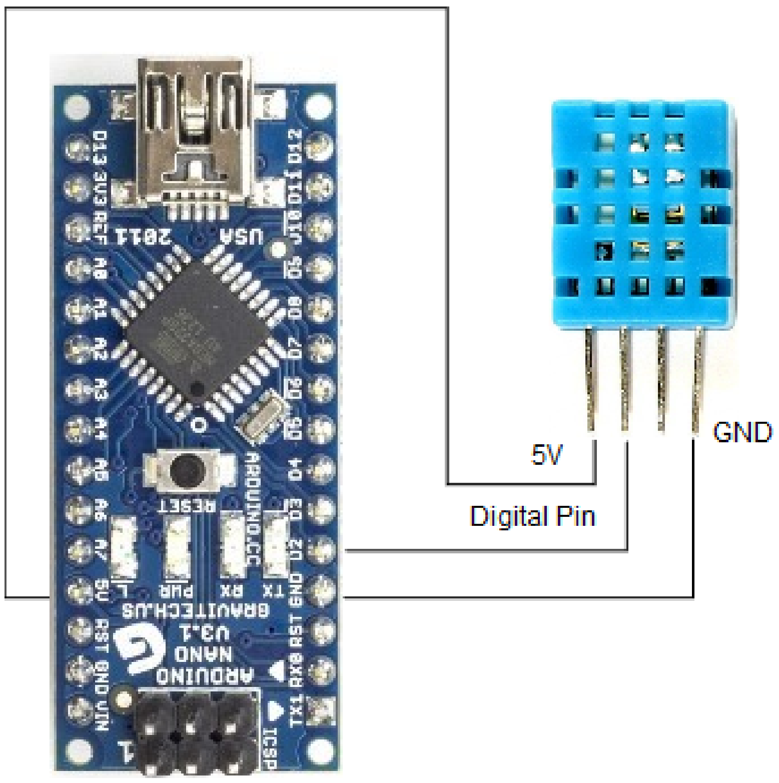


Figure 4: DHT11 Temperature and Relative Humidity Sensor Scheme.[2]

The principle of GP2Y10 optical dust sensor's function is that inside it a LED flashes on a light sensor and depending on how much dust/smoke is in the air, the measure it takes is high in the clean air and low in polluted. For the sake of easier interactions with the sensor, the output voltage is reversed in IC - low in the clean environment and high if it's polluted.

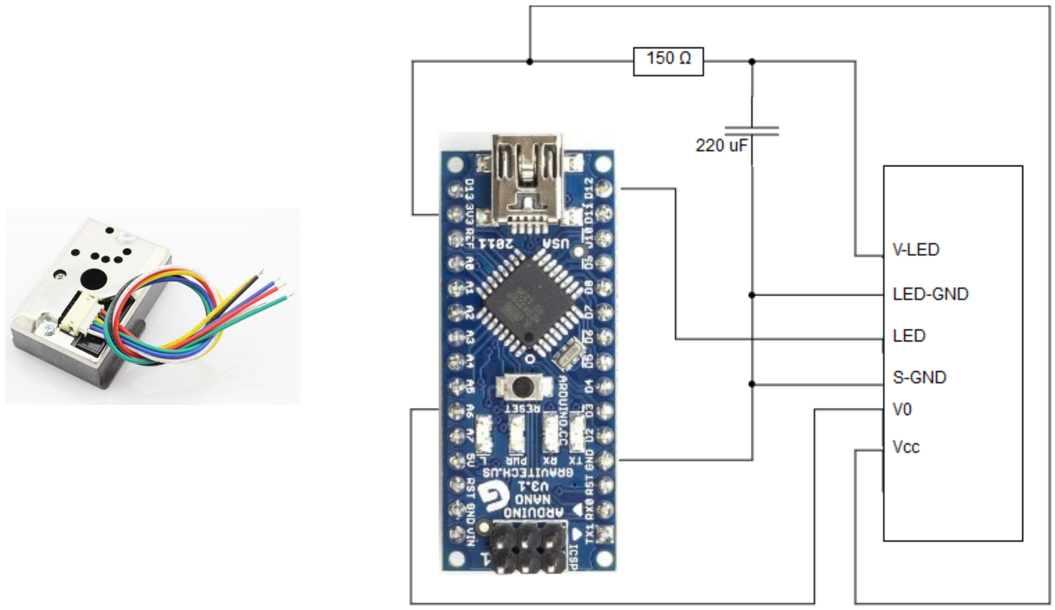


Figure 5: Left - GP2Y10 Optical Dust Sensor, right - its connection scheme.[3]

MH-Z19 carbon dioxide sensor exploits the CO2's property of being less transparent to light in infrared spectrum than other normal atmospheric gases. The measuring principle is that it sends some light of two different frequencies, but with the same intensity, and using a light sensor compares the two measured intensities. Basically, if the air sample has none CO2 at all, measures are going to show some close, if not the same numbers, but if there is a lot of CO2 - measures will be contrasting because CO2 would absorb a noticeable amount of infrared light's energy.

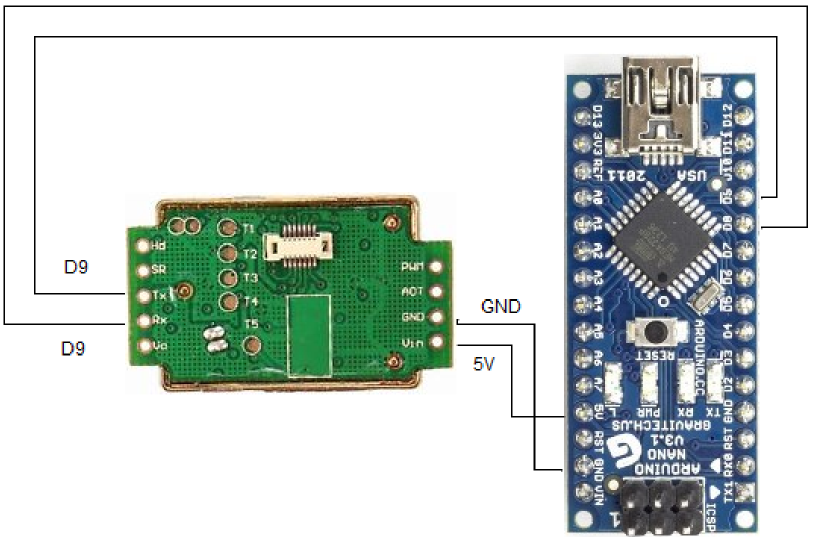


Figure 6: MH-Z19b CO2 Sensor Scheme.[4]

4.2 Measure Process

For all the sensors the measurements follow an algorithm as follows:

1. Arduino sends a request to a sensor, which means sequences that are a little different for each sensor
    - For DHT11 request sequence is a simple 18ms log.0 followed by 40us of log.1 and then waiting for response
    - Sharp dust sensor is the simplest one control command-wise, for the only command it uses to start a measure is “power up the LED and wait some time until the light sensor is saturated so that measures can be taken”
    - The MH-Z19 CO2 sensor uses digital input in a form of nine consecutive bytes:
        - 0xFF - Start Byte
        - sensor number - unless multiple sensors are connected to the same control pin it’s 0x01
        - command byte - not all the range of commands is tested yet; intended use - as is written in the manufacturer’s datasheet - mostly suggests the use of 0x86 (read CO2 level) and 0x87 (calibrate zero-point)
        - Five empty bytes 0x00
        - Checksum of first 7 bytes
2. Sensors take their measurements and send back responses
    - DHT11: as were mentioned before, it measures the resistance for both its elements, translates those into digital format and sends this information back to Arduino in form of five consecutive bytes: humidity integer data - humidity fractional data - temperature integer data - temperature fractional data - checksum. As is relatively obvious from the names - the sensor takes the decimal numbers of humidity and temperature and sends them in binary form with two bytes each - for the parts before and after the decimal point; the last byte is the checksum and it is there to control if all the data was transferred correctly - if it does not equal to a sum of the first four bytes, it means the data sent is unreliable and it is advised to re-run measurement.
    - Dust sensor uses just the “LED power on” from the first step to conduct the measure, so next step is to use Arduino analog input to measure the signal coming from the light sensor.
    - MH-Z19 follows the order to do the measure, in which it shines two different wavelengths through an air sample on a light sensor, and compares the resulting intensities. After that it sends back the same amount of data as were in the request - nine bytes, arranged as follows:
        - Start Byte - 0xFF
        - Command repeated back, usually 0x86
        - Two bytes of CO2 ppm value, for high and low parts
        - Four bytes that are not officially documented and not yet entirely understood - just like most command bytes
        - Checksum

3. Arduino reads the responses from sensors, if needed - makes calculations like the conversion from analog input voltage $[V]$ to $[mg/m^3]$ for dust sensor, or just conversion of the data to decimal numbers from binary, and sends said numbers in JSON format together with sensor identifiers over the serial port.

4.3 Data Processing

After conversion of raw data from sensors, Arduino sends it all over the serial port in JSON format, here is the example of such messages for all sensors:

- {“DHT”: {“humi”: 49, “temp”: 22.7}} - which means that DHT11 sensor measured relative air humidity at 49% and temperature at 22.7 degrees Celsius
- {“Dust”: 0.01} - dust sensor’s measure of 0.01 mg/m3 of dust, and
- {“CO2”: 600} - MH-Z19 sensor’s measure of 600 parts per million of carbon dioxide

All of these are caught by Raspberry Pi using a script, that reads JSON messages, sorts them in terms of which sensor they belong to and sends them to the corresponding combination of:

- Network address, in this case - localhost at 127.0.0.1
- Network port of Statsd input, here - 8125
- Metric name, here - apostle602.edu.dust, .temp, .humi, and .co2
- Metric type, here - ms, meaning histogram

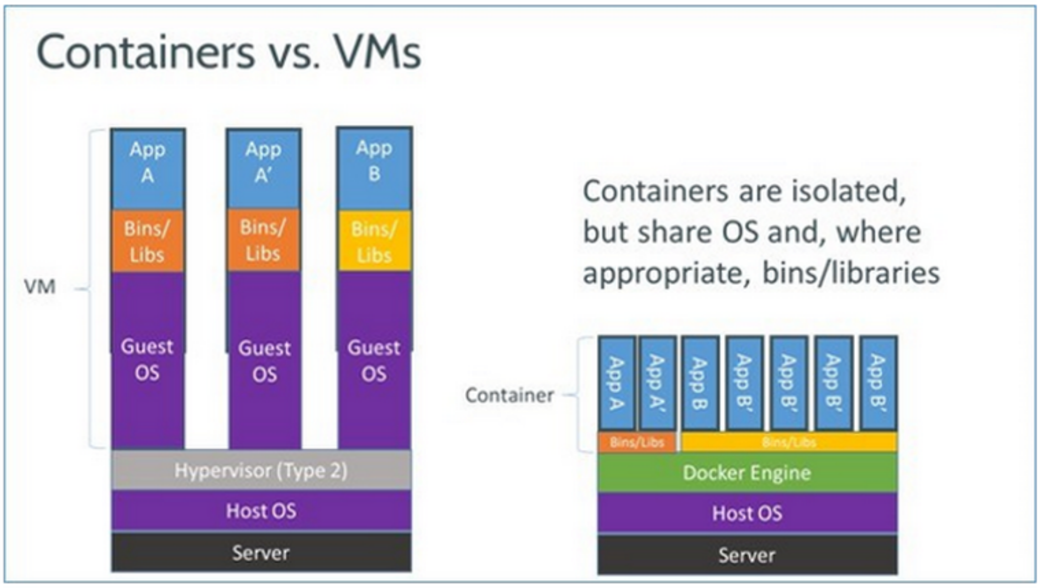


Figure 7: Visual comparison between Docker and Virtual Machines.[5] [6]

Said script is placed in a container called reader-sender for Docker environment. Docker is a container platform often used for enterprise needs, for example, so that it would be easier to transport some parts of work-in-progress code between different systems: instead of making virtual machines it can be used to create a container image that is a "lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings". In this work it has been mostly utilized as a platform to run a container with the pre-configured combination of Graphite[7], Grafana[8], and StatsD[9], and - to make all of this configuration run smoothly both in function and design - the sensor output reader has been put in a neighboring container. As a pleasant addition, Docker can not only be used to create these containers, but also to make them run on system startup, and to reboot a container if some part of it is found to be down for some reason.

The "reader-sender" container first sends all the data flowing through it to the localhost:8125 port, designated to Statsd. It is an extension needed for a correct metrics aggregation: StatsD can be configured to process these metrics in some specific ways, e.g. one can set it up to different types of accounting metrics that came in inside a small time interval, like the maximum value, minimum, sum, average, etc.

Within the framework of arrangement in question, it may be not exactly needed, because, for the purpose of making graphs with exact data values coming in one time a second at most, we do not need all of these conversions, but it allows making different graphs if the need would suddenly arise, and it also would make it easier to attach more sensors, especially those with higher rates of measures, or to just add some more reliable graphs for some system information, e.g. processor activity or analysis of connections from outside - all without the need to recompile all of the metric collection system.

StatsD is set up to send the aggregated data every 5 seconds to Graphite, to put it simply - a database to keep all the collected data, taking relatively little disk space, and keeping up with all the timestamps. Here are some problems with Graphite though: if two metrics come in with the same timestamp it will only save the last one, and if for example it is set up to aggregate these metrics once in 10 seconds, which is a default and is mostly enough of an accuracy for any purpose, and, say, in this time period it has ten metrics of nine ones and one of 200, it will count average as a middle point between the maximum and the minimum values, here - 100 instead of 20.9 that is usually understood as an average in these conditions. Again, it may not be a problem in this particular sensor set up, but in an environment where it would make a difference - StatsD is exactly the tool that can get rid of this kind of imprecision.

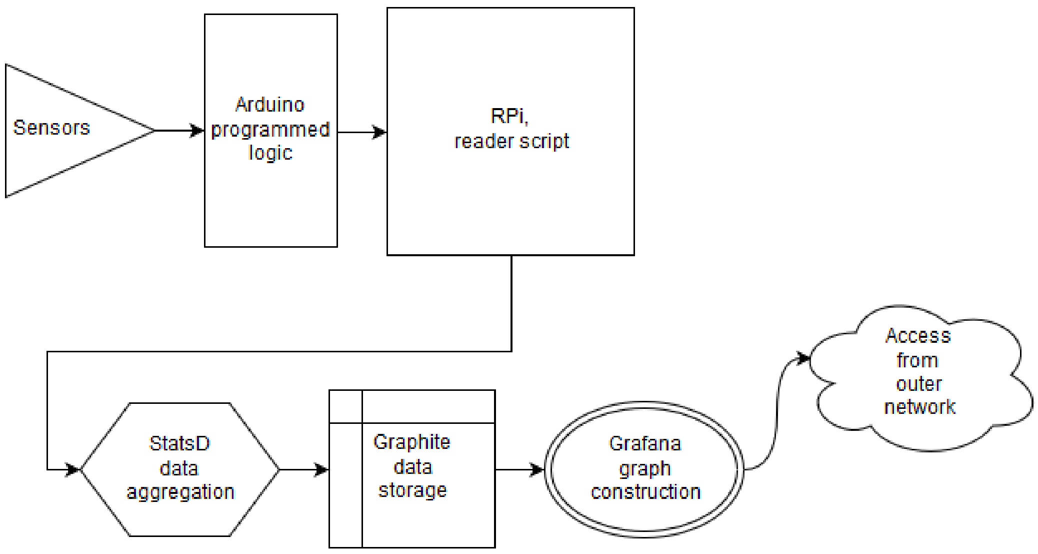


Figure 8: Logic Schematic.

After all the metrics were correctly aggregated by StatsD and saved to Graphite, Grafana can access this data to combine it into easily readable and, what can be important if it is intended to be operated by an end user instead of just the system administrators, appealingly designed graphs, other types of panels and dashboards. Grafana offers a wide range of varied graph types and design options, which are mostly not included in the basic build without extensions, but by default, these panels are provided: [10]

- Graph - with possibility of including multiple metrics, grouping, number of design options and three options on X-axis:
  - Time - with grouping data by time
  - Series - instead of time the data is grouped by series
  - Histogram - bar chart, shows how many data points fall in some specified ranges

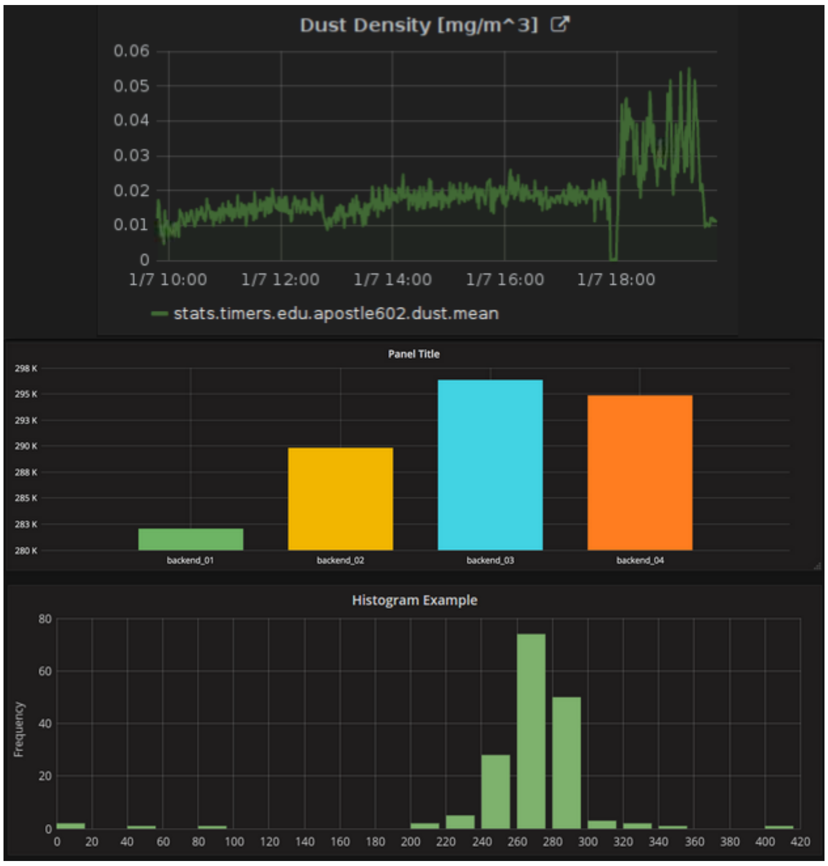


Figure 9: Examples of Graph panel.

- Singlestat - as is obvious from the name, is a panel that allows insertion of a single series, so it then shows the last known value with a number of Singlestat-specific functions like min, max, delta, etc. Includes these options:
  - Simple value output - may have three color zones depending on set up thresholds.
  - Spark Line - functions like a simpler version of a graph, as a way of seeing the historical data without accent on its accuracy.
  - Gauge - to give a clear picture of how high a value is in its context.
  - Value to text mapping - as a way to translate some values to easily readable message, e.g. if the value is critically high, instead of a number it can show "Danger" message.

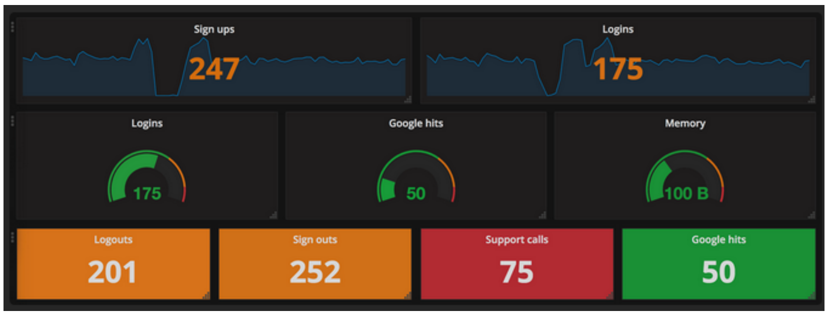
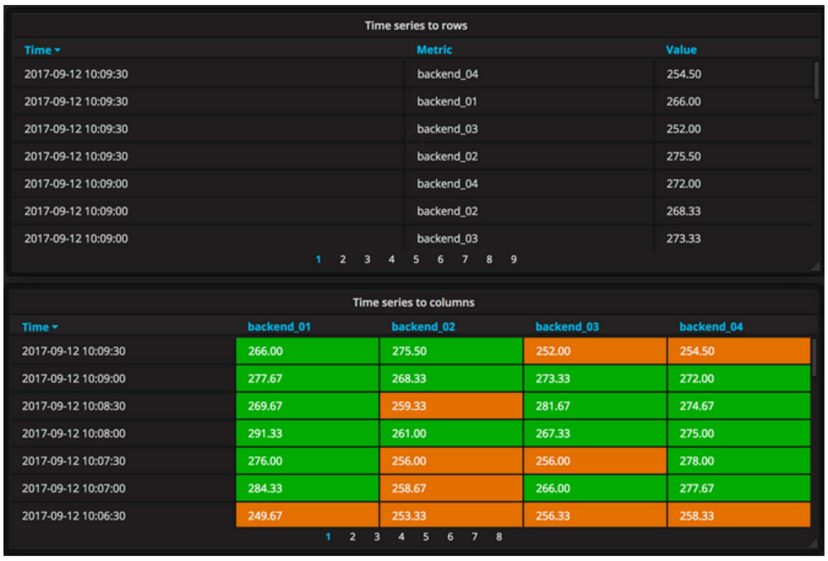


Figure 10: Examples of Singlestat panel.

- Table - has two main uses:
  - Transformation of time series to columns, which is pretty much like a graph with accentuation on specific values instead of their fluctuations.
  - Time series aggregation, which is very similar to what is Singlestat used for, with the difference being the possibility to put in multiple metrics in a single table for ease of comparison.



Time series to rows

Time ▾	Metric	Value
2017-09-12 10:09:30	backend_04	254.50
2017-09-12 10:09:30	backend_01	266.00
2017-09-12 10:09:30	backend_03	252.00
2017-09-12 10:09:30	backend_02	275.50
2017-09-12 10:09:00	backend_04	272.00
2017-09-12 10:09:00	backend_02	268.33
2017-09-12 10:09:00	backend_03	273.33

1 2 3 4 5 6 7 8 9

Time series to columns

Time ▾	backend_01	backend_02	backend_03	backend_04
2017-09-12 10:09:30	266.00	275.50	252.00	254.50
2017-09-12 10:09:00	277.67	268.33	273.33	272.00
2017-09-12 10:08:30	269.67	259.33	281.67	274.67
2017-09-12 10:08:00	291.33	261.00	267.33	275.00
2017-09-12 10:07:30	276.00	256.00	256.00	278.00
2017-09-12 10:07:00	284.33	258.67	266.00	277.67
2017-09-12 10:06:30	249.67	253.33	256.33	258.33

1 2 3 4 5 6 7 8

Figure 11: Examples of Table panel.[?]

- Heatmap - a representation of histogram over time, where each time slice represents its own histogram.

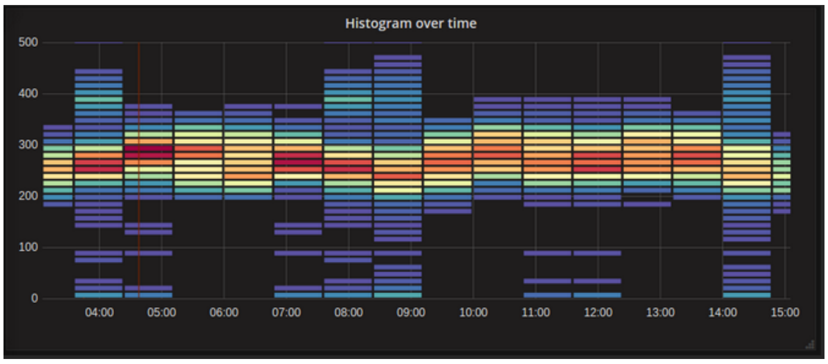
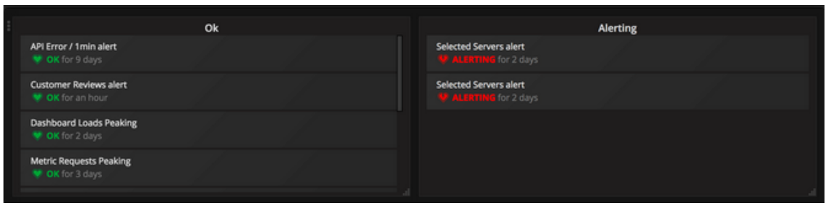


Figure 12: Example of Heatmap.

- Alert List - allows to display dashboard alerts. The list can be configured to show current state or recent state changes.



Ok

API Error / 1min alert
OK for 9 days

Customer Reviews alert
OK for an hour

Dashboard Loads Peaking
OK for 2 days

Metric Requests Peaking
OK for 3 days

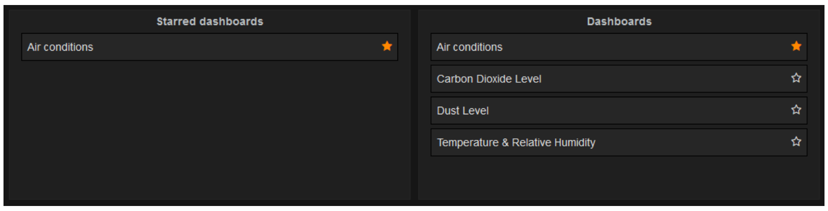
Alerting

Selected Servers alert
ALERTING for 2 days

Selected Servers alert
ALERTING for 2 days

Figure 13: Example of an Alert List.

- Text - self-explanatory, allows insertion of any text or HTML code, e.g. description for other panels in the dashboard or a picture.
- Dashboard list - a place to keep links to all relevant dashboards, that may be very useful for navigation in case there is a considerable number of dashboards on different subjects.



Starred dashboards

Air conditions ★

Dashboards

Air conditions ★
Carbon Dioxide Level ☆
Dust Level ☆
Temperature & Relative Humidity ☆

Figure 14: Example of a Dashboard List.

5 Results

5.1 Measures

Here are some results that can be seen in Grafana, in the Air Conditions dashboard.

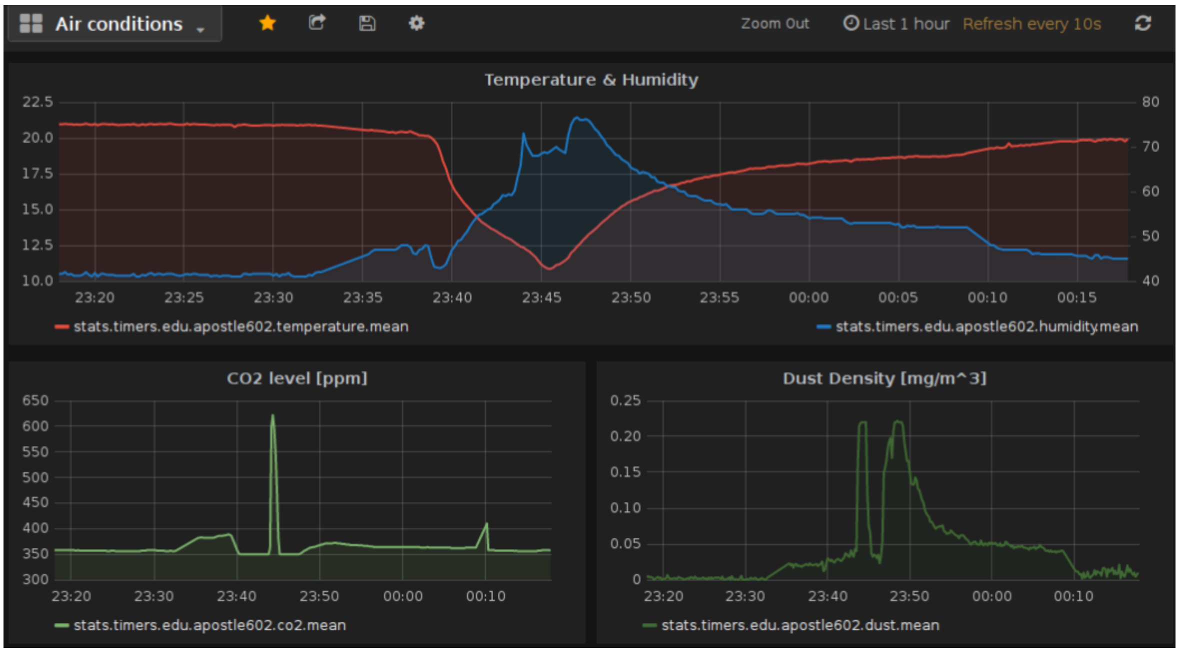


Figure 15: Air Conditions Dashboard, All sensors, one hour

For this example of a one-hour air monitoring, a short trip outside has been performed, cca. 23:35 to 23:45. Peaks on CO2 and dust sensors resemble moments when tobacco smoke has been exhaled directly over said sensors.

From the Temperature/Humidity graph, a conclusion can be made, that DHT11 sensor used in this work is not intended to detect any immediate temperature fluctuations, because it takes a considerable amount of time to cool down and heat up to the environment temperatures. Also it can be seen as a reminder that it's the relative humidity and not the absolute value, because even though its graph shows even quick fluctuations, the overall read value slowly changes according to the temperature readings.

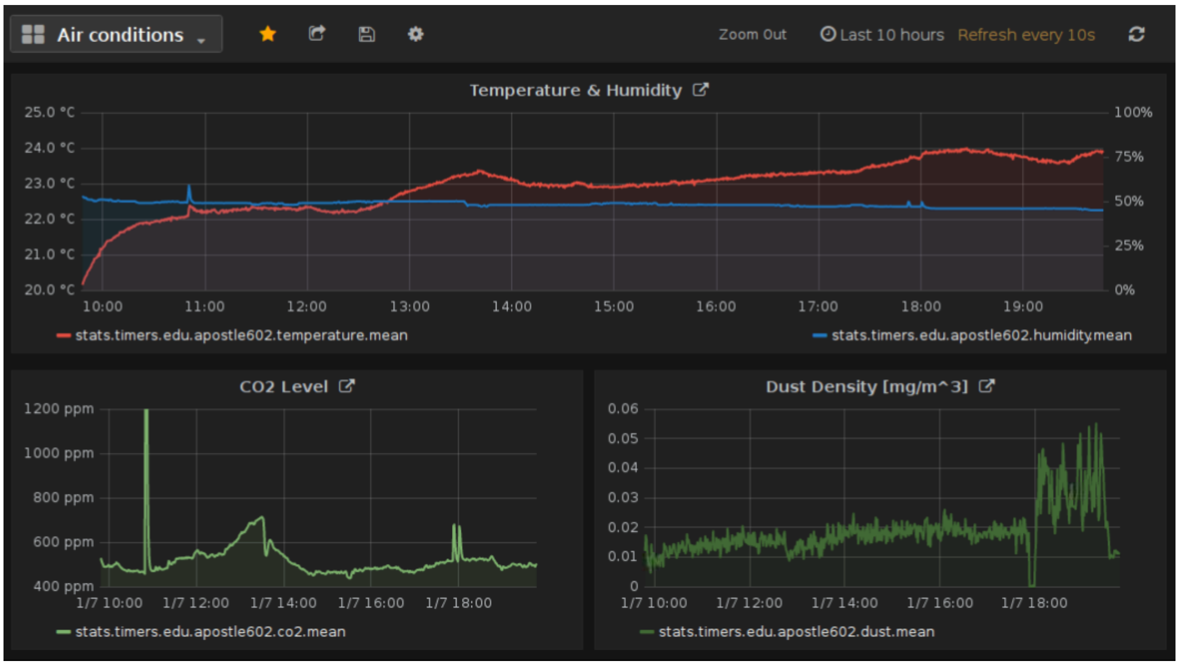


Figure 16: Air Conditions Dashboard, All sensors, ten hours

Example of ten-hour home air monitoring. Peaks on CO2 and Temp/Humidity around 10:50 resemble direct exhalation over sensors, time from cca. 13 to 14 hours indicates indirect human physical proximity to sensors, peaks on all the sensors around 18:00 resemble cooking in the same room.

5.2 Implementation of Plug-and-Play

Arduino is set up to continually check all three sensors, without the check if any of them is plugged in. The option of disconnection of any given sensor is implemented simply by not sending any data from a measure if it did not yield any results - it is obvious with the DHT11 and MH-Z19 sensors, for they are both digital and the absence of their response is not hard to distinguish from some specific response.

Although this was harder to implement with an analog Sharp Dust sensor: when Arduino performs a reading on an analog pin, it creates some voltage on it, that it then mistakes for an input and eventually sends erroneous data. There was an attempt to fix that by comparing the mentioned analog pin reading to another, free analog pin - the difference between these showed up to be relatively constant. Alas, it is not a perfect way, for it does not entirely cull the erroneous data, and if the actual environmental dust value will appear just right for these pins to have similar readings, it will not show the measured data.

As for the connection of the sensors back again, the most important part is that nothing has to be reprogrammed for the system to start reading their evaluations, but still there may be some problems:

- The DHT11 sensor did not show any kind of errors after disconnection/connection to the running system, it just stops sending any JSON messages when it's not in the circuit and continues to send them after being put in again.
- CO2 sensor did show inconsistent results: in half of all test cases it just continued to send its data as if nothing happened, other times it started to send something un-

readable. This has proven to be completely fixed by turning Arduino off for the time needed to connect the sensor.

- Dust sensor actually showed better results with connection than disconnection - no problems were detected except for that it had to make a couple measures to go up the actual reading value.

So it would be recommended to turn Arduino off before any connection, if not for another problem: RPi works on Linux and it has its own way of assigning names to USB ports - the first one operational gets number 0, next gets 1 and so on. For reasons that were not entirely understood, sometimes RPi gave the USB port used by Arduino with number 1, and the reader script is not intended for use on any other number than 0, so no readings were sent to aggregation.

5.3 Insertion of a new sensor

In existing implementation, there are two ways to add a new sensor:

- Connect it to Arduino, and append an evaluation code to the existing Arduino program, or
- Connect it directly to RPi through pins, and create a new bash script for sensor evaluation.

After that the output of both options should be fed into StatsD with reader-sender script, that also has to be rebuilt so as to include the new JSON tag.

It should be noted, that by default, without any extra dongles RPi does not have any analog pins, so if a sensor has an analog output, it can only be connected to Arduino, or to an extra dongle for RPi.

6 Conclusion

The basic cost of such system is somewhere around 3300 czech crowns without the power bank or power adapter.

It can be lowered mainly by the use of a simpler SBC, for example an older model of Raspberry Pi, like the model 2, that is probably powerful enough to perform the same.

Also of course it has to have some casing, preferably so that the circuitry connections are not visible at all.

References

[1] CSc. Prof. Ing. Miroslav Husák. *Senzory v elektronice a informatice - Přednašky*. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2013.

[2] A dht11 class for arduino. https://playground.arduino.cc/Main/DHT11Lib.

[3] Standalone: Sharp dust sensor. http://arduinodev.woofex.net/2012/12/01/standalone-sharp-dust-sensor/.

[4] Measuring co2 concentration in an apartment with mh-z19. https://geektimes.ru/post/278178/.

[5] Docker homepage. https://www.docker.com/.

[6] What is docker and why is it so darn popular? http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/.

[7] Graphite homepage. https://graphiteapp.org/.

[8] Grafana homepage. https://grafana.com/.

[9] Etsy/statsd home on github. https://github.com/etsy/statsd.

[10] Grafana documentation. http://docs.grafana.org/.