Journal of
Sensor and Actuator Networks

MDPI

Article

Low-Cost IoT: A Holistic Approach

Augusto Ciuffoletti

Department of Computer Science, University of Pisa, 56127 Pisa, Italy; augusto@di.unipi.it

Received: 5 April 2018; Accepted: 4 May 2018; Published: 8 May 2018

Abstract: The key factors for a successful smart-city project are its initial cost and its scalability. The initial cost depends on several inter-related aspects that cannot be designed and optimized separately. After the pilot deployment, scaling-up takes place only if the cost remains affordable: an initial financial support may induce dependencies from technologies that become unsustainable in the long period. In addition, the initial adoption of an emerging technology that fails to affirm may jeopardize investment return. This paper investigates a smart-village use case, the success of which strongly depends on the initial cost and scalability, exploring a low-cost way for Internet of Things (IoT). We propose a simple conceptual framework for cost evaluation, and we verify its effectiveness with an exhaustive use case: a prototype sensor designed and tested with its surrounding eco-system. Using experimental results, we can estimate both performance and cost for a pilot system made of fifty sensors deployed in an urban area. We show that such cost grows linearly with system size, taking advantage of widely adopted technologies. The code and the design of the prototype are available, so that all steps are reproducible.

Keywords: Internet of Things (IoT); smart village; low-cost; REST/HTTP; WiFi; virtual clock

1. Introduction

While several metropolitan cities have already successfully launched experiments for environmental monitoring (like Barcelona [1] or Amsterdam [2] in Europe), there is a declared interest to extend the experience to small towns, in the framework of wider projects [3–6]. A challenging aspect of such initiatives is their long-term sustainability, especially in the transient from a pilot phase to large scale deployment, as pointed out in [2]. This condition, per se critical, is definitely compromised if external funding is suspended, and, unfortunately, sometimes a pilot project happens to lose interest when its success is demonstrated.

Therefore sustainability has to be evaluated in the early stages of long-term perspective projects: designers and administrators should bear in mind that a small community is unwilling to support an expensive service, once the cost of success falls on its shoulders. Low cost, so long as system performance is adequate for the task, allows taking full advantage of the efforts needed to launch the project. This is why the low-cost keyword is relevant, and its importance is in fact demonstrated by the frequency with which this feature appears in Internet of Things (IoT) literature.

Despite being so crucial, this concept is often left implicit, or imprecisely defined, considering only a few components in the overall financial impact. Consequently, it may happen that an unexpensive sensor depends on an expensive infrastructure, or that low costs are actually reached, but only in the large scale.

In this paper, we consider the most restrictive situation, that of a small size project without perspectives of economies of scale, and we define all the costs entailed in its realization. We keep into account and exploit the presence of a collaborative social community, which significantly contributes to lowering the project cost: this happens when users actively participate in its implementation and

support. The community expects from project realization a better quality of life, but gains also the ability to share experiences and data with others, possibly with a financial return.

We will use the term smart village, mediated from [3], to indicate the framework hosting the project. In short, the smart village is a small community, with limited resources, but the solid intent to improve its control on environmental resources, including air, water, energy, roads, parking lots, etc. From such experiences, more than from generously financed pilot projects, others may obtain useful hints, thus making the IoT really improve our lives.

The Cost of an IoT Project

The financial investment needed to implement an IoT project is split into several components; each of them must be evaluated in order to pursue an overall low-cost strategy. Let us indicate five relevant items:

- several sensors/actuators, that collect data, pre-process them, and implement the requested actions,
- a network that connects the sensors to data aggregators,
- a service infrastructure that aggregates and renders the information,
- salary for the developers and maintainers,
- energy requirements.

The task of minimizing the cost of a single component is not challenging, since prices are steadily decreasing in this sector: it is more complicated to find a compromise that makes the whole project affordable, which is the aim of this paper. Let us explore the dependencies among these components, summarized in Figure 1.

The first two items in the above list are strongly related, since the sensor needs to interface with the network infrastructure. At this moment, two Low Power Wide Area (LPWA) technologies are in the process of reaching the market: the Long Range technology (LoRa), which operates on unlicensed frequencies with a proprietary scheme, and the Narrow Band IoT, integrated in the Long Term Evolution (4G-LTE) standard, which operates on licensed frequencies. Both of them promise to become low-cost carriers for IoT data, but, at this moment in time, they are quite expensive in terms of equipment and deployment or licenses, and introduce a further risk related with their success and diffusion. Looking at more established technologies, we find the WiFi, with a solid experience in the creation of urban networks, and the 3G cellular network, which entails relevant per-unit costs.

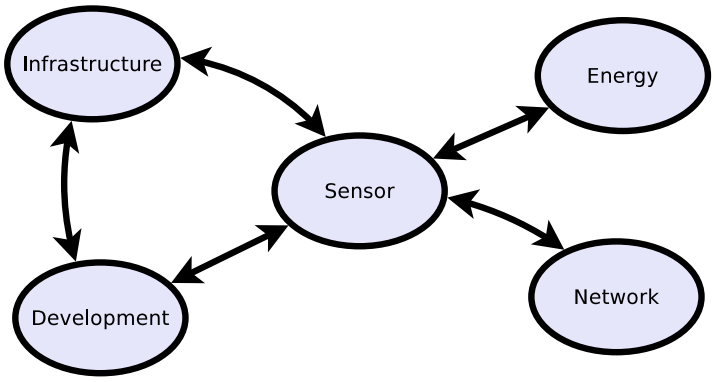

Figure 1. Cost components and their relationships.

Considering the processing infrastructure, the diffusion of specific IoT cloud services simplifies the task of deploying the infrastructure, and delegates maintenance, power, and logistics to the provider under a pay-per-use policy. A cloud deployment is elastic by nature, the initial cost being possibly null, smoothly increasing with the size of the project. There is currently a number of providers that offer attractive user interfaces, one of them being the popular ThingSpeak, but it is also possible to

take advantage and compose raw cloud resources, like data storage and web servers, that are available at lower prices.

The utilization of cloud services implies a connection from the network of things to the Internet, and we find several infrastructures whose business model encapsulates cloud components, like SigFox.

Concerning development costs, we argue that they are tightly related with the quality of higher education: they are lower when the project is based on popular technologies, which new generations learn in Information and Communication Technology (ICT) courses. On the contrary, cutting edge technologies have high costs. Using a popular technology, a collaborative community can contribute to further lower the development costs: consider the case of a sensor implemented as a high school course competition, or as the result of a crowd-sourcing initiative.

Energy consumption is the last component in the list, and the less relevant. Although our target community does not ignore green precepts, it also does not want to incur relevant costs in order to have devices that consume 0.5 mW instead of 50 mW. Nonetheless, it is relevant to evaluate this aspect, in order to understand whether the device depends on the electrical grid, or may operate for months on batteries (which, by the way, have a carbon footprint), or gathers energy from the environment (solar, wind, bumps etc.), thus incurring a significant initial cost.

Finally, although costs are an issue, system performance needs to be adequate for the purpose. The design may trade-off certain features, but there is a kernel of functions that needs to be present. In other words, it is important to minimize the cost, but keeping the service useful.

This paper focuses on IoT affordability, trying to define a viable entry point for a small community with a limited budget. For each of the components listed above, we are going to indicate the tools that help to start an IoT project with limited resources.

It is a valuable and original contribution, since most papers address one aspect of the design, disregarding the others. Instead, we want to undertake a holistic approach for the definition of a viable strategy for smart villages.

2. Low-Cost Internet of Things

Let's start our discussion from the networking infrastructure, which is the cornerstone of an IoT system; the current trend is to deploy a dedicated infrastructure for IoT data, using Low Power Wide Area Network (LPWAN) technologies, in addition to the ones already existing for voice and human oriented Internet. Although LPWANs are rather stable from the technical point of view, on the commercial side, strong contrasts are arising between hardware and service providers. Therefore, our target community is not willing to take part in this game, in the fear of losing investments in a technology that soon becomes outdated and unsupported.

The alternative is to use established technologies, like WiFi or 3G. From the technical point of view, they have very little in common, but both exhibit the basic capability to send or receive small pieces of data, being powered only when needed.

The advantage of 3G is that it does not need the deployment of an infrastructure to connect a device to the Internet, since the local telecom provider is in charge of it. Although such service comes at a cost, many 3G plans are emerging that prepare the market for the advent of LTE-IoT products, with a cost of a few dimes per month. The cost of a 3G interface is in the range of $20, and Single Board Computers (SBC), with integrated 3G capabilities are available for around $60.

In contrast, a WiFi deployment needs a network of access points (AP), a complex and expensive operation. However, WiFi is a serious competitor of 3G in a smart village perspective, since the same infrastructure is reusable for other services, like an urban WiFi service for shops, schools, and libraries. In some cases, then, the service is already in place, and the IoT network can simply take advantage of it. Another point in favor of WiFi is that interfaces are currently sold for a few dollars. Although we may expect that, in the near future, LTE-IoT interfaces will reach similar costs; as a matter of fact, the same budget can buy ten 3G interfaces or two WiFi hotspots with fifty interfaces. Depending on whether the target application is sparse or dense, today either one of them is the best choice.

Among LPWan technologies, LoRaWan offers an attractive trade-off between WiFi and 3G, coupling a wide range reach, with a non-proprietary infrastructure. A comparison between LoRaWan and other LPWan technologies is found in [7], cost analysis included, and a critical view of LoRaWan is in [8]. There are three reasons that suggest to defer the adoption of such technology:

- popularity: the development is more expensive, since there is less experience and support for this sophisticated technology;
- shareability: the LoRaWan, as any LPWan infrastructure, is dedicated to IoT, while WiFi provides other services;
- stability: LoRaWan technology is rapidly evolving and has strong competitors [9]. It is therefore exposed to improvements that are going to make existing equipment obsolete.

The price of LoRaWan devices is still slightly higher than that of WiFi: in the conclusive section, we sketch a comparison between LoRaWan and WiFi, and discover that the difference of hardware costs is an extra 50% for LoRaWan. Since popularity, stability and price are all exposed to change, in the future, a LPWAN approach may become attractive for low-cost IoT systems.

To understand the advantages of a popular, shareable, and stable technology, consider the case of a deployment in a supportive community, where families and shops with a public or private Access Point are happy to adopt a mote, at the cost of a weekly battery recharge, or with a negligible contribution in power and bandwidth. Although this opportunity depends on the connectivity of the surroundings, there is no doubt that adopting a popular technology has social aspects that increase the level of participation and contribution.

Sensor technology is tightly related to software production costs: if a popular development tool is available for the sensor device, standard skills are sufficient to write the driving code. This is favorable on the side of salary and development time. If the devices exhibit a common structure, we obtain a similar advantage since designers accumulate experience, and one design is readily reused in others.

Two low-cost candidates with popular coding tools are the boards based on AVR chips (Atmel, San Jose, CA, USA) (including the Arduino), and on the ESP8266 (Espressif Systems, Shangai, China). Both are programmable using the Arduino Integrated Design Environment (IDE), which is successfully used in schools of any degree (and even in a CoderDojo Kata). Other popular candidates are not equally appropriate for the task, for reasons related with costs or development tools. For instance, the Raspberry Pi, as well as other similar Single Board Computers (SBC), has a cost, power consumption, and management complexity that dis-encourage its adoption as a low-cost sensor device; another example is Particle Photon, which is not as popular as the Arduino family, and is currently more expensive. STM32 based boards are a concrete alternative, but not as popular as the Arduinos.

While choosing an IoT-specific cloud service, one needs to take into account the risks of lock-in and price escalation. A user can be locked in an IoT service in various ways that extend those found in cloud computing. Here are listed four commercial policies that tend to lock-in an IoT business:

- data lock-in: when data can't be easily ported to other providers and technologies, as with any other cloud provision,
- technology lock-in: when the data repository is immersed in an ecosystem of smart services that permeate the application,
- device binding: when the IoT device is bound to use only a given data repository,
- code binding: when the code for the IoT device is bound to use a given code repository.

Every cloud provider uses a different mix of the above commercial strategies, and it is difficult to foresee which is more dangerous for a smart village. In order to protect the investment, probably device and code binding should be avoided at all costs. Technology lock-in may be attractive, since it helps to simplify the application, thus reducing development costs. Data lock-in is not relevant if the application does not make use of historical data, like in a simple sensor/actuator loop control; otherwise, the designer should check that periodic data download is feasible.

Price escalation occurs when the implemented project becomes popular and needs more resources to grow: this may jeopardize scaling. In some cases, the price curve may be more than linear as the project leaves the small scale of a pilot deployment: cloud services are often provided for free in early stages. Even in the favorable hypotheses of a linear price growth, the community should be prepared to an increment of service cost, after it becomes really useful.

To cut the costs, it is possible to create an infrastructure using general purpose cloud services, which are usually less expensive than specialized ones: for example, a PaaS web server as user interface and data upload end-point, and a cloud storage as data container. The infrastructure costs are lower than those of an IoT specific service, but development costs rise.

A further alternative consists of the creation of a server, either in the cloud or not, running one of the available open-source IoT servers: ThingSpeak is one of them. In this case, the community needs to pay for the server (either on premises or in the cloud) and for its maintenance.

The server should also provide data access not limited to raw measurements, but encapsulated in web applications and services. Although it is possible to commission this task to external developers, it is preferable that data are left open to those that contribute to the project, so that apps emerge directly from the users. This option is viable only if the competence to create such applications is present in the community: the adoption of cloud services with well known, standard interfaces helps.

The rest of this paper is organized as follows: after a survey of works that directly address low-cost IoT systems, we introduce a case study that exemplifies and gives concreteness to the design principles defined above. A fundamental step on this way is the definition of a benchmark, a set of features that we want to be present in our system: such step states the rationale for the technical options that are finally introduced in the prototype, and allows future comparisons with other solutions that aim to reach the common goal of an affordable IoT architecture.

3. Brief Survey of Research Papers on Low-Cost IoT

This section is dedicated to a brief survey of recent IoT papers that have been selected because they mention low-cost as one of the requirements. As we will see, this attribute is usually applied only to one aspect of the problem, disregarding the overall cost, and frequently not fully justified.

Most of them come from developing countries, which demonstrates that IoT is considered as a way to improve their quality of life, filling the gap with developed countries.

The applications considered in the selected papers may be divided into two families: urban mobility, and management of primary resources, like water or energy.

The first family may be further divided into two sub-streams: traffic monitoring, and public transportation management.

In the first sub-stream, Ref. [10] investigates traffic and road surface control using a compact device that embeds a number of sensors, including a GPS receiver and an accelerometer. The paper does not mention experimental results or implementation details to produce the device. Its cost is estimated of $30, and the study comes from the USA.

With a similar target, Ref. [11] uses branded devices to fetch traffic characteristics, which are made available to drivers. They expect a favorable impact on air pollution and economy. There is no explicit statement about cost, though the authors claim to pursue a low one. The study comes from India.

In Ref. [12], the authors design a system that helps drivers find a free parking lot. Low-cost is not addressed in this paper, but the design actually adopts low-cost technologies for hardware and networking that are complementary to those explored in this paper: namely, an STM32 (STMicroelectronics, Geneva, Switzerland) for the SBC, and Narrowband IoT (NB-IoT) for networking. The study comes from China.

The public transportation sub-stream is represented by [13], which uses WiFi sniffing techniques to count people at bus stops and thus make a more effective use of coaches. The authors claim a low-cost result, but the sensor device is a Raspberry Pi, a rather expensive, though powerful, SBC, with a relevant power consumption. On the topic of power consumption, Ref. [14] quantifies the

difference between the Raspberry and Arduino-like boards. The authors do not evaluate the cost of the device, but illustrate experimental results of an on-field experiment. The paper comes from the USA.

There is another family of IoT applications where low-cost requirements are frequently found: it aims to improve primary resources management.

The quality of the air is fundamental for an urban environment, and we have found three papers on the subject:

- Ref. [15] uses a BeagleBone SBC (similar in concept and cost to a Rasperry Pi) equipped with sensors and a GPS receiver. Low-cost, mentioned in the abstract, is not quantified. The work comes from India.
- Ref. [16] is the paper that, in this survey, is most aware of aspects related to the cost. The authors describe the prototype design and evaluate energy consumption, showing its dependency from the target application. The Web infrastructure is based on an on-premises server, while sensor networking uses WiFi. The work comes from Chile.
- Ref. [17] uses the GPRS infrastructure to deliver data obtained by low-cost sensors. The authors tackle the goal of using low-cost sensors, which dominate other costs in these kinds of applications. The paper marginally describes the infrastructure needed to collect, process, and present the data. The work comes from Serbia.

Low-cost IoT solutions are also proposed for the management of supply networks.

The authors of [18] design a device for billing water consumption and to detect leakage and losses: they do not introduce a new service, but aim at reducing the cost of an existing one. Several ways to deliver data collected by sensors are explored, including handheld devices, GPRS and wireless Local Area Networks (LAN). The design is from an Indian institution.

The electric grid is considered, with a similar purpose, in [19]. Energy consumption is measured by an Arduino-like SBC and forwarded to a nearby Bluetooth device, where data are collected and possibly uploaded to a database. The intent is to reduce costs and improve accuracy of conventional devices. The authors are from Egypt.

In [20], the authors want to improve the management of electrical energy coming from several renewable sources by controlling dynamic parameters of distribution lines and accumulators. The paper gives little detail of the infrastructure used, and concentrates on end user applications and sensors. The authors are from an Indian institution.

The reliability of the power supply grid is the target of [21]. An Arduino SBC is used to capture signs of overload from electrical transformers using low cost sensors. Data are uploaded to a cloud database using a wireless LAN, and records are kept for analysis. Also this work comes from India.

An application that is receiving increasing attention consists of a dense deployment of low-cost, non-intrusive devices to monitor the safety of resorts areas, like parks and historical sites. Video capture and WiFi sniffing allow a fine grain analysis of people around, but are not respectful of privacy. Instead, coarse grain devices, like sound monitors and infrared sensors, gather relevant information without compromising privacy and at a lower cost. In [22], the authors give the details of an IoT system based on such kinds of sensors. The paper describes how data is gathered and filtered to obtain significant data, and analyze the aspects of energy harvesting to power sensor devices. The authors are from Singapore.

In summary, none of the papers gives sufficient details of all the aspects that contribute to the cost. Likewise, none of them gives enough information to reproduce the solution. An analytic cost evaluation and design reproducibility are the focus of this paper.

4. A Case Study: Arduino, WiFi, and ThingSpeak

Our case study is based on three cornerstones: Arduino, WiFi, and ThingSpeak. Around them, we have implemented a complete solution, and we want to evaluate its limits and strengths: in this section, we explain why we consider them as low-cost; in the next section, the solution is matched against a benchmark application.

We start considering the sensor/actuator, which is the heart of an IoT solution. This component is replicated, and the number of replicas is a measure of the size of the system; thus, the impact of its design is also multiplied. A flexible design may bring an economy of scale in the development, since software components are reused.

Our choice for this component is Arduino, which is indeed the name of a family of development SBCs (Single Board Computer) built around an Atmel AVR. We discard Arduino boards that have an integrated WiFi interface: in fact, a survey reveals that such boards have a cost that is considerably higher than the sum of the parts. In addition, the adoption of an integrated architecture restricts design options, and often binds to external libraries and firmware. We conclude that the adoption of two distinct boards, one for the SBC, another for the network interface, has a favorable impact on costs, while keeping the design open and flexible.

For our purpose we need a board in the Arduino family that is suitable for deployment, not markedly oriented to development. The Arduino Mini-pro board meets such requirements, and costs a few dollars: it hosts a Atmel Mega328p MCU, the same as the popular Arduino Uno, and program development can be carried out with the same Integrated Design Environment (IDE). The cost of a Mini Pro is lower than that of an Arduino Uno: on eBay, $2 and $4, respectively. Power consumption is also lower, since the 16*U*2 ISP programmer is not included in the board (the schematics are available on [23,24] respectively). Pinning is standard Dual In Line (DIP), which is compatible with all sorts of prototype boards and printed circuit design tools. It can be programmed using specific pins that allow the connection of the external ISP programmer. Its size is six times smaller than that of the Arduino Uno: 6.5 cm^2, instead of 35 cm^2, allowing a reasonably compact device.

Considering development costs, the Arduino IDE is extremely popular, which makes its utilization easy and supported by a huge number of know-how sources and projects. The development process with the Mini-pro is not distinguishable from that of the Uno, so that an Arduino-grown developer feels at ease.

Another viable alternative, offering higher performance at a lower cost, are STM32 based boards ([25]). However, their IDE is far less popular, and the learning curve is steep. Here, we need to trade-off development costs for processing power, and we want to privilege the former.

The WiFi technology has the advantage of a ubiquitous distribution, with low-cost devices for the infrastructure, and affordable network interfaces. In the introduction, we discussed the possibility of engaging the community itself in the provisioning of the WiFi infrastructure, using the connectivity provided by shops and homes, when they have sufficient coverage. This option makes it expensive to extend the reach of the IoT beyond city limits, since reaching the Internet from the sensor device may require a dedicated Access Point, or a broadband device. Therefore, the use case is applicable only within city limits, excluding the countryside.

The interface between the sensor and the WiFi network is implemented using an ESP-01 device, a 4 cm^2 board based on the ESP8266 processor.

The ESP-01 exposes a serial interface, controlled by a hardware UART, from which it receives commands that control the embedded WiFi interface. They are encapsulated in an AT syntax, that mimics the one of old telephone modems: for instance, the command `AT+CWJAP="mywifi","friend"` is used to join the WiFi network the Extended Service Set Identification (ESSID) of which is `mywifi`, using the password `friend`. In case of success, the response is an `OK`, with an additional content depending on the invoked command.

Data transport is performed with a sequence of two or more commands: the first one specifies length and target of the operation, and is followed by a sequence of frames containing the data. The initial command prepares the buffers and arranges the transport level connection, while the following send or receive data.

Notably, the standard socket abstraction is not present, and the coordination between SBC and WiFi interface is not straightforward: a library is needed to simplify the task of application developers. The alternative consists of using another library that provides the standard socket interface (like [26]).

However, we observed that such library uses most of the available memory resources, thus requiring a more powerful SBC: in the trade-off between development and device costs, we opted again for the latter. The penalties are that coding a library is more expensive, since it requires deeper professional skills, and that using a non-standard tool-set also complicates the application developer task.

To attenuate the latter aspect, the library provides a simplified application level interface: the transport layer is still accessible, but application layer short-cuts are also available. To select the concepts to implement in the library, we observe that a number of application protocols are emerging that explicitly address IoT—for instance, the Message Queue Telemetry Transport (MQTT) protocol (see [27]). We prefer to adopt HTTP: although less powerful, it is more stable, known, and adopted than any other application level protocol. Security measures are easy to implement in an HTTP infrastructure, since a proxy server can secure IoT data exchange between the encrypted WiFi network, and a remote HTTPS server, without implementing the Secure Socket Layer (SSL) protocol on the sensor. In addition, IoT cloud servers frequently adopt an HTTP interface conforming to the Representational State Transfer (REST) paradigm.

Our case study uses the ThingSpeak service. Its offer includes, besides the REST interface for data upload, a rich set of satellite services that simplify the application design, but expose to technology lock-in. The designer should be aware of that, since the service is rather expensive, and costs grow linearly after a generous initial free plan. To contain the costs, the ThingSpeak server may be moved on premises, since its source code is available: this may attenuate the impact of technology lock-in, but has a cost related with equipment and maintenance.

Figure 2 illustrates the architecture for our case study: hardware components are vertical stacks, and horizontal layers correspond to the Internet ones, from network to application. The leftmost hardware component is the Arduino, with library and application code discussed above. The former uses a serial interface with the ESP-01 to perform operations at transport level. The shortcuts provided by the library enable a direct interface from the application sketch to the ThingSpeak channel implemented on the server. No intermediate components are interposed along the data path, in contrast to edge computing or fog approaches. As discussed in [28], such concepts are quite popular in IoT literature, and are introduced to allow integration of components with limited computing and communication capabilities, which becomes mandatory in the presence of networking infrastructures with an extremely low bandwidth, as in the case of LPWAN. However, they have a significant impact on hardware costs, and in design complexity. In this paper, we show how to avoid edge computing in a smart village application, despite the limited computing capabilities available on sensor nodes.

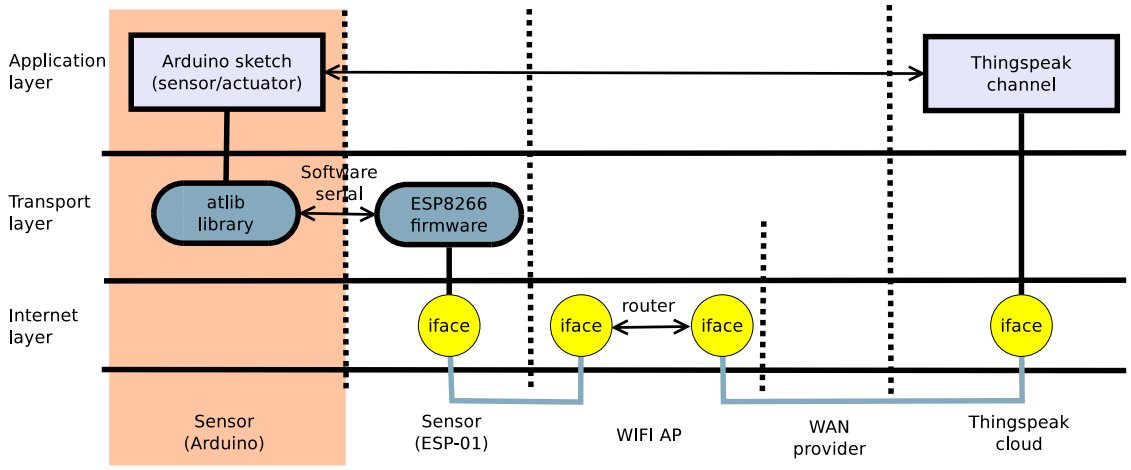

Figure 2. The proposed architecture envisions application level communication between the sensor and the service provider, thus reducing development costs.

Now that the architecture has been defined, we proceed to define a benchmark, a kernel of capabilities that characterize our application, before checking them in a prototype implementation.

5. A Benchmark for Prototype Evaluation

The definition of a benchmark allows understanding the limits of an implementation, and gives the basis for comparison. In our case, the benchmark consists of relevant features that are implemented on the sensor, but that depend on the overall architecture. We found a number of such features during the design of the prototype in [29], and the benchmark we propose focuses on three of them: clock synchronization, upstream/downstream communication, and power saving.

Clock synchronization is mandatory to coordinate the operation of several components, for instance to control traffic lights in a street. We may want that traffic light B becomes green ten seconds after light A: in that case, an accurate time reference is needed, but its resolution is coarse, in the order of a second. The benchmark requires the availability of a time reference, and measures its accuracy.

Communication consists of two sensor-centered activities: posting new values (upstream), and downloading commands (downstream). The sensor controls update operations, and can implement resource saving policies, for instance powering the network interface only when needed. Communication in the other direction is controlled by the upstream device, and a protocol is needed to preserve sensor resources. A solution is based on a polling mechanism controlled by the sensor, which periodically queries for the presence of pending commands. Another consists of turning on sensor receivers at predictable times, thus avoiding expensive transmissions, but requires synchronization. The simplest alternative consists of coupling upload and download operations. All the above alternatives share a drawback: the sensor-actuator loop has a long time constant. This prevents real-time reactions, unless power saving requirements are significantly relaxed.

Here, we prefer to consider power as a valuable, and possibly scarce, resource. Thus, the benchmark requires the implementation of power saving policies, especially in the WiFi interface management for communication and clock synchronization, but regards a timely reaction from the system as not relevant. An evaluation of power consumption is also needed to adopt a cost-effective power supply.

The implementation of the above core functions hits against Arduino limits, since clock drift is significant, the available memory is limited, and sleeping degrades response. Regarding memory, it is split into two parts: program memory (30K bytes), and data memory (2K bytes). Real limits are even lower, to take into account that memory requirements change at runtime, in response to dynamic memory allocation events. As a matter of fact, the Arduino IDE rejects programs that exceed 60% of the available memory.

To determine the feasibility of the core functions, we evaluate the storage footprint of the benchmark application that implements them. The energy footprint is measured using as a reference the capacity of a pack of three rechargeable AAA batteries, 800 mAh nominal. Given a voltage supply of 3.6 V, the available energy is 2.9 Wh. Dividing this figure by the measured autonomy of the device in hours, we obtain an easily reproducible approximation of power consumption.

Program development complexity is difficult to evaluate, but such figure is important to understand the effort needed to solve a specific use case. The count of statements is used for this purpose, but we know that the library has development costs higher than applications.

In summary, the benchmark requires implementing three relevant features on the sensor/actuator:

- a clock with a bound accuracy with respect to a standard reference,
- an HTTP/REST client with GET/POST capabilities,
- functions to switch to low power mode the WiFi interface.

The evaluation of their implementation is split into four metrics: clock accuracy, storage footprint, program complexity, and power consumption. Regarding the storage footprint, we need to take into

account that data acquisition and command actuation, i.e., the business code of the sensor, are not included in the benchmark.

5.1. A Prototype and Its Evaluation

The prototype system [30] is composed of the architectural components in Figure 2: a sensor/actuator, a communication network, and a cloud service infrastructure. The development process targets the sensor: such component has a high impact on project sustainability since it is replicated, and therefore exposed to scale-up issues. This component also determines energy consumption.

The sensor/actuator is implemented by assembling an Arduino Pro Mini and an ESP-01 on a 5×7 cm prototype board: the two components are mounted on a socket to be easily replaced and reused, but a more compact assembly is easily done. Figure 3 is the electronic layout, and in Figure 4 a breadboard prototype.

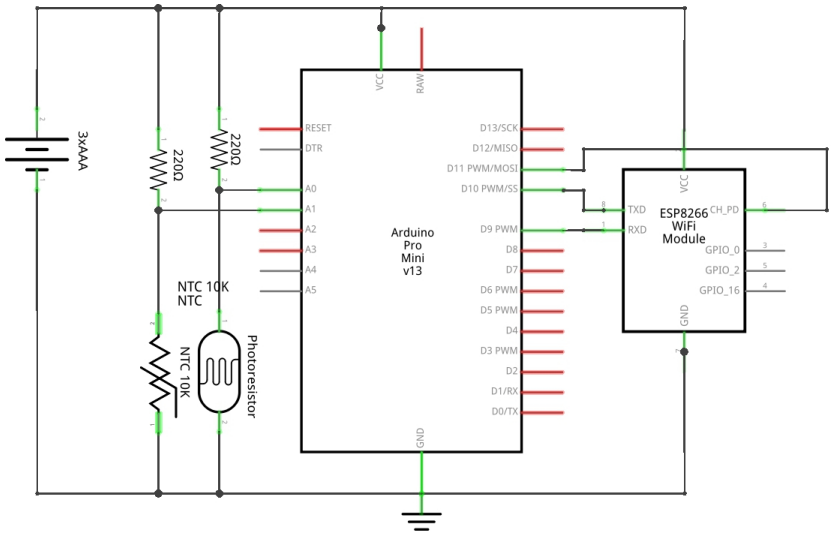

Figure 3. Schema of the sensor used in the evaluation prototype.

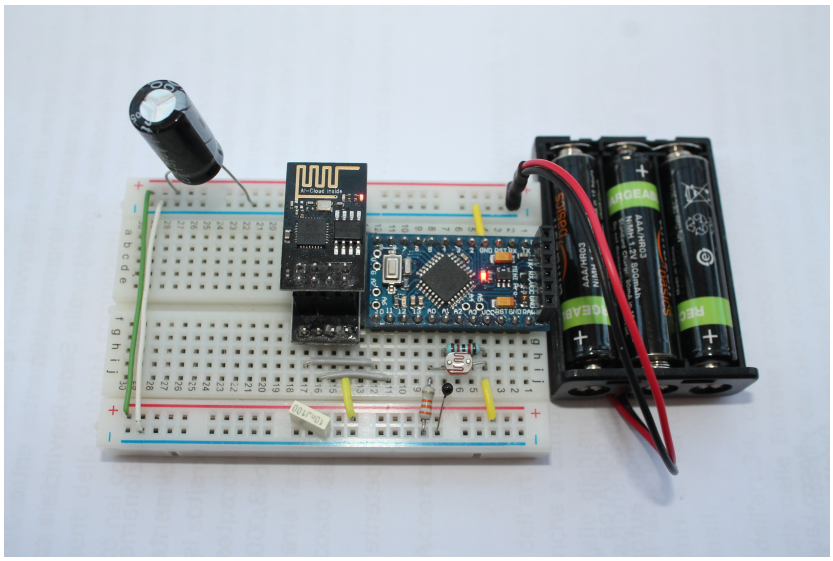

Figure 4. An early prototype of the sensor, mounted on a breadboard.

The two components have the advantage to be available as Commercial Off-The-Shelf (COTS) devices, but still have the limits of development boards: in order to be user friendly, they carry

hardware, like power LEDs and stabilizers, that is useless when the device is in operation. This aspect leaves some space for an economy of scale, once the experimental phase is complete.

Power supply is from three AAA rechargeable batteries, the nominal capacity of which (800 mAh) is used to quantify power consumption. Given that battery voltage is compatible with the requirements of both boards (3.3 V), there is no power adapter between them.

The Pro-mini has an 8 Mhz clock, and is connected to the ESP-01 through a serial interface with a baudrate of 57,600. On the Arduino side, the interface is implemented in software, while the ESP-01 uses its own hardware UART (see Figure 3).

The Arduino library that simplifies the utilization of the serial WiFi interface [31] implements a class that encapsulates the ESP-01 device. It offers two distinct kinds of methods: a wrapper for AT commands, and application utilities.

The application utilities provide:

- a REST interface to interact with a ThingSpeak server,
- the capability to enable or disable the operation of the ESP-01 device,
- an accurate virtual clock.

Note that Arduino libraries already exist for the three tasks (the mentioned WiFiEsp [26], the ThingSpeak library, and the NTP library). However, the Arduino can't support the three of them together. Instead the ad hoc library provides the desired functions, and leaves enough resources for business code.

Communication with the ThingSpeak server follows the HTTP, and is therefore split into a Request and a Response. The Request is implemented using a raw AT command to establish a TCP connection, and one or more calls to send the fragmented HTTP Request to the server. The Response is processed by another library function that parses the input from the server, and reports about the success of the operation, returning the Response message body when present. Such functions make a greedy usage of buffers, and 250 bytes are sufficient for the task.

Two methods are devoted to switch on or off the ESP-01 device. The function that implements the switch-on, besides raising the enable pin of the ESP-01 board, also takes care to process the output from the ESP-01, since the device automatically joins the AP using the credentials stored in the flash memory. This latter feature of the ESP8266 helps to limit the size of the sketch, which does not implement the WiFi join procedure.

Clock synchronization capabilities are split into two library functions: one that periodically gets in touch with an NTP server and computes the current drift and offset of the hardware clock, another that returns the UTC time by correcting the local clock value using such drift and offset. The function does not use the NTP protocol itself, but the time service available on port 37. The primary advantage is a very compact code on the sensor. In addition, it is easy to implement the time service on an on premises server equipped with a GPS receiver.

The sensor is connected to the Internet using a commercial WiFi ADSL router as Access Point (AP). The association of a sensor with the AP occurs every time the ESP-01 on the sensor wakes up to perform network operations, like delivering data or synchronizing the virtual clock.

The IoT infrastructure is provided by ThingSpeak: using that service the sensor is able to upload measurements, download commands and configuration parameters, send and receive tweets, trigger actions based on time and data. All such services are reached through the REST API provided by ThingSpeak.

5.2. Measurements and Results

The Arduino Mini-Pro runs a benchmark application [32] that verifies and stresses the three features anticipated in Section 5: clock synchronization, REST interface, and power saving. As business code, every two seconds, it samples the value of two sensors, an NTC and a photo-resistor that are visible in Figure 4.

Every fifteen minutes (or 900 s) the sensor connects to the WiFi AP and performs the following tasks:

- it updates the virtual clock parameters with a TCP connection to an accurate time server,
- it uploads eight measurements, included the average temperature and lighting during the period, to a ThingSpeak channel with an HTTP POST,
- it downloads a command from a ThingSpeak TalkBack resource with an HTTP GET.

When the three operations are complete, the WiFI interface is switched-off until the next communication round.

The first result we obtain is that hardware is sufficient for the task: the application uses 42% of the 30 KBytes available for programs, and another 42% of the 2 KBytes available for the memory. As a reference, the official ThingSpeak library requires 99% of the program memory to implement a single feed (using an Arduino Ethernet shield), while the WiFiEsp library uses 43 percent of the available memory for a single GET, leaving little space for further functions.

The complexity of the development task is condensed in the number of lines required by the library, which is 300 lines long, and of the program, which is 200 lines long. The library is an expensive but reusable part of the project, which is split into nine methods that contribute to the definition of a class. Each method is typically twenty lines long, except for the HTTP response parsing method that is 100 lines long with a complex structure.

The main program has a straightforward loop structure that runs the sampling task and the periodic connection. This latter is implemented by another function, which performs a sequence of three connections, respectively for clock synchronization, data upload and command download.

We consider that development is split into subtasks—the library and the applications—each of them manageable by a single moderately trained C++ programmer with Arduino experience.

To evaluate the performance of the prototype, we run the benchmark described in the previous section, measuring:

- the time spent performing the communication tasks,
- the roundtrip time with the NTP server,
- the measured drift,
- the time error measured at each re-synchronization,
- the power supply.

The benchmark application stops when batteries run lower the brown out threshold, which is 2.9 V: this allows estimating power consumption.

Drift measurement is a critical task that encompasses sensor and network functions: its success proves, alone, that system performance is stable in time.

The drift is computed using the formula below:

$$drift = \frac{utc_1 - utc_0}{ck_1 - ck_0}, \tag{1}$$

where utc_1 and ck_1 represent the current UTC time and the current clock, and utc_0 and ck_0 represent the same figure at system startup. Since all figures have a one second resolution, the resolution of the time service, a drift estimate is computed using an Exponentially Weighted Moving Average (EWMA) with gain 4. The result is shown in Figure 5.

The EWMA has a significant effect only during the initial transient, when the two terms of the ratio in Equation (1) are very close.

The drift stabilizes around 3000 ppm, corresponding to approximately 2.7 s every fifteen minutes. We note that, unless compensated, such a drift prevents the utilization of the internal clock for synchronizing distributed activities within the desired one second window. Using the drift estimate, it is possible to implement a virtual clock [33] with the required precision. The effectiveness (and

correctness) of drift compensation is shown in Figure 6, which represents the difference between the value of the compensated virtual clock, and the time value received from the NTP reference.

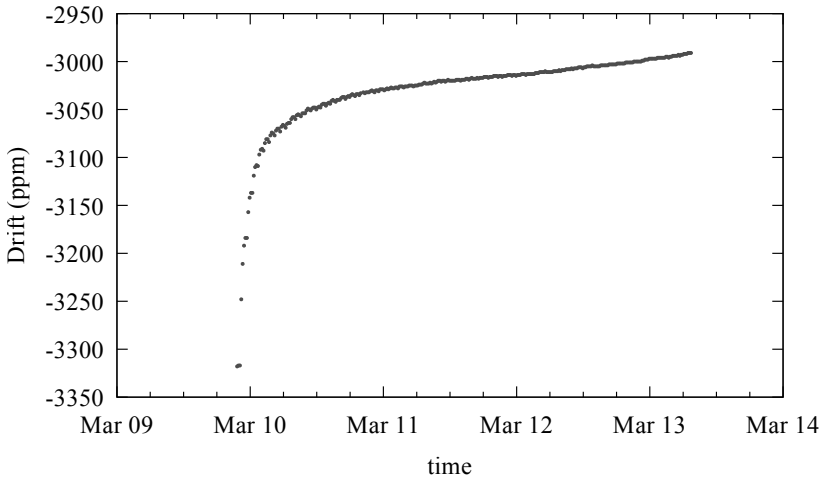

Figure 5. Drift estimate during the experiment.

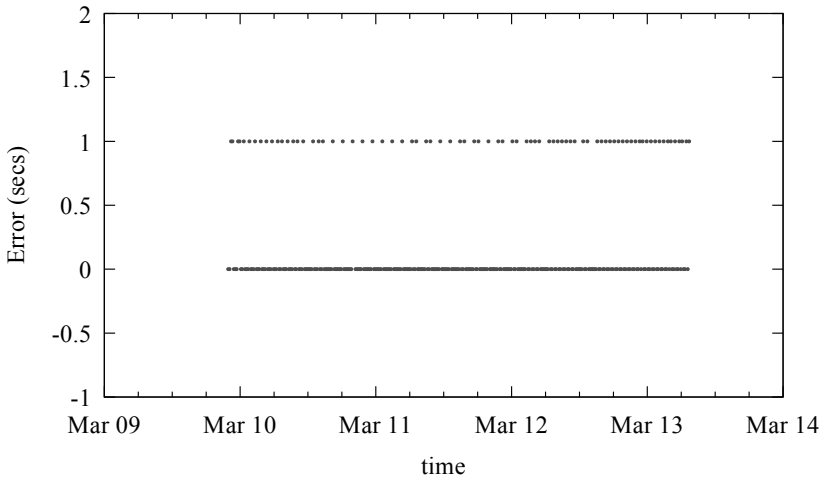

Figure 6. Time offset between the internal virtual clock and the reference time server, as measured during clock synchronization.

The first value of the series exceeds the requirement of a one second precision, since drift estimates requires at least two clock readings to work. However, after the first value, the error remains confined in the [0, 1] range.

The roundtrip measurement refers to a Stratum 0 Network Time Protocol (NTP) server at the USA National Institute of Standards and Technology (NIST), and shows that network communication is stable. It successfully recovers after an apparent network problem (see Figure 7): in that case, the roundtrip time jumps to 5 s, without compromising the application. The average roundtrip is 419 ms, and the standard deviation is 290 ms, as shown in the frequency distribution in Figure 8, with an intriguing bi-modal pattern. The average roundtrip time dis-encourages an effort towards accuracies significantly lower than one second. With the provision of a local time server, the roundtrip time could be reduced of one or two orders, allowing an accuracy in the range of tens of milliseconds.

The real-time capabilities of the prototype are limited in the tens of seconds, since the update of the Thingspeak channel on the public cloud requires a time narrowly centered around 15 s (see

Figure 9). The presence of an on premises Thingspeak server, or a multi-tier architecture [34], would certainly improve the real-time capabilities, at a cost.

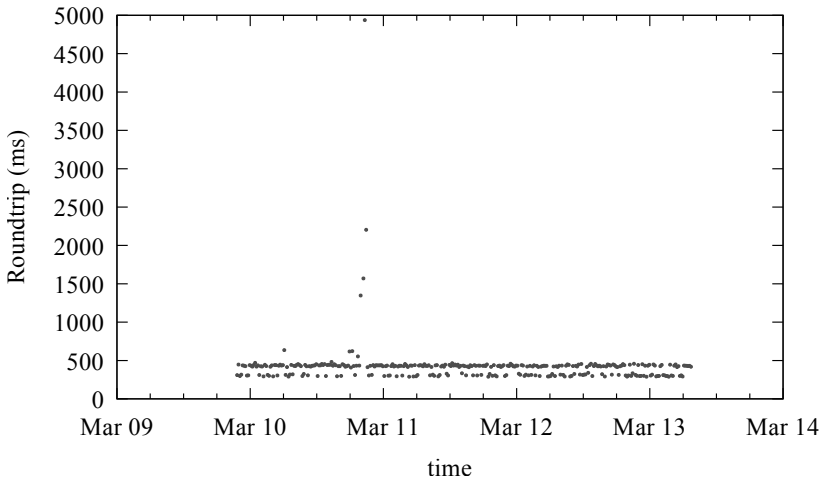

Figure 7. Round-trip time with the reference time server, as measured during clock synchronization.

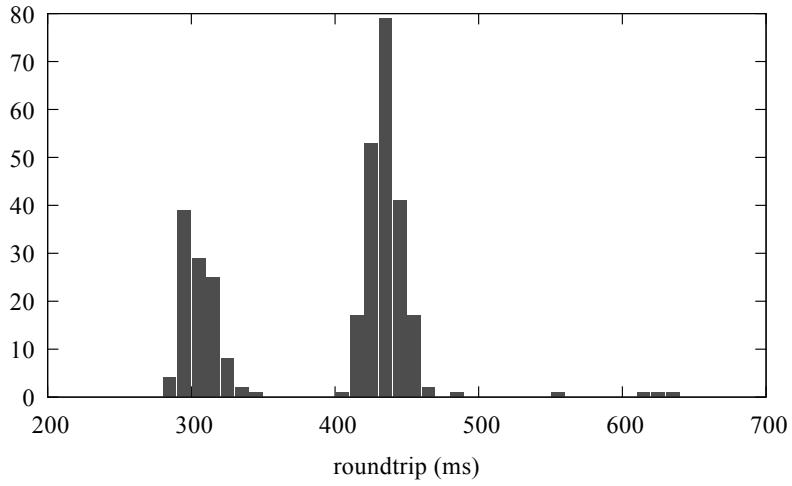

Figure 8. Frequency distribution of round-trip times in Figure 7.

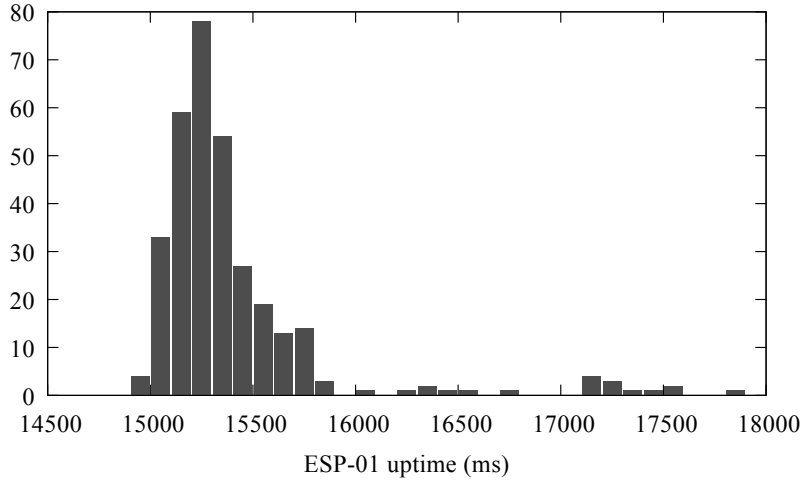

Figure 9. Frequency distribution of the time spent with the WiFi interface switched on during each round.

We infer power consumption from the capacity of the batteries, nominal 800 mAh, and the duration of the experiment, that lasted 327 rounds of 15′ each, corresponding to 81.75 h (more than three days). The rate between battery capacity and experiment duration returns an estimate of the average current, 9.8 mA, and of the average power consumption, that, assuming an average 3.5 V supply (see Figure 10), is 35 mW. Gathering such power from the environment is not difficult: for instance, a postcard sized solar panel (2 W/6 V) might be sufficient for the task.

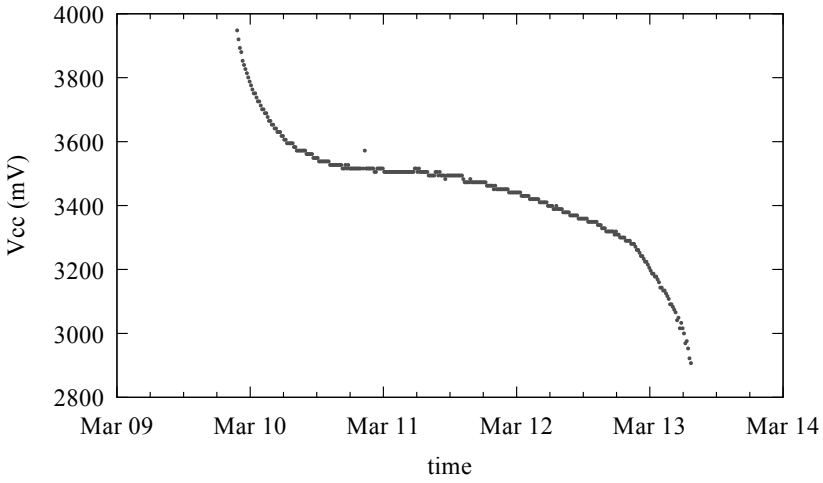

Figure 10. Variation of the power supply (Vcc) during the experiment.

Understanding the distribution of power consumption between the two devices, the Arduino Mini-Pro and the ESP-01, helps to devise a power-saving strategy. To this purpose, we run a further experiment, reducing the period between successive communication sessions from 15 to 5 min. Since the overall power consumption can be split into two components, one with the network device in sleep mode, the other with the same device awaken, we can write:

$$C_{acc} = t_{sleep} \times I_{sleep} + t_{wake} \times I_{wake}, \tag{2}$$

where C_{acc} is the accumulator capacity, t_{sleep} is the time spent with the WiFi device off, t_{wake} the time spent with the WiFi device on, I_{sleep} and I_{wake} the supply currents in the two cases.

The linear system obtained using the values from the two experiments is the following:

$$\begin{aligned} 80.33\ \text{h} \times I_{sleep} + 1.42\ \text{h} \times I_{wake} &= 800\ \text{mAh}, \\ 56.84\ \text{h} \times I_{sleep} + 3.25\ \text{h} \times I_{wake} &= 800\ \text{mAh}, \end{aligned}$$

which is solved with

$$\begin{aligned} I_{sleep} &= 8.1\ \text{mA}, \\ I_{wake} &= 104.3\ \text{mA}, \end{aligned}$$

which are reasonably consistent with datasheets.

Considering the benchmark execution—with a period of 15′—the average supply current of the whole device is 9.8 mA, with a constant contribution of 8.1 mA from the Mini-pro, corresponding to 83%. This candidates the Mini-pro as a target for the application energy saving policies.

There are plenty of ways to reduce power consumption of an Arduino [35]. One way is to enter the *powerDown* state for a definite amount of time. In our use case, the sampling period is two seconds long and the sampling operation is rather fast. It is a favorable situation, and power saving is in fact significant, but the solution can not be generalized.

To quantify the power consumption when the Arduino is in the PowerDown state, we run another experiment with the same period, but entering the PowerDown state during the time between sampling operations. The 800 mAh accumulator lasted 165.5 h (nearly one week), with an average power consumption of 17 mW. To compute the power consumption shares, we write the following equation:

$$C_{acc} = t_{powerdown} \times I_{powerdown} + t_{sleep} \times I_{sleep} + t_{wake} \times I_{wake}, \quad (3)$$

where we have added a new term for the time when both units are in the power saving state. Using the results of the previous step, and the data from the new experiment, we obtain:

$$800 \text{ mAh} / = 159.79 \times I_{powerdown} + 2.91 \text{ h} \times 8.1 \text{ mA} + 2.80 \text{ h} \times 104.3 \text{ mA}, \quad (4)$$

and we obtain a value for $I_{powerdown}$:

$$I_{powerdown} = 3.0 \text{ mA}.$$

Since the average current during the experiment is now 4.8 mA, $I_{powerdown}$ corresponds to 60% of the overall power consumption. Both boards are in power saving mode, and datasheets claim that their power consumption should now be far less than 1 mA , while we estimate 3 mA. We conclude that parasite components significantly contribute to $I_{powerdown}$: in fact, we have two power LEDs and one unused stabilizer on the boards. We conclude that producing an ad hoc board without development-oriented components would further reduce power consumption.

6. Conclusions

We have shown how a smart village can launch an IoT project with a limited initial investment and little or no external funding.

To reach this conclusion, we have considered the whole project as split in tightly interdependent components. We have found a consistent set of low-cost solutions, one for each project component, and we have verified that the overall system is able to support a benchmark application. We do not claim it to be the optimal solution, but one ensuring low cost and the presence of a defined set of features. The approach is driven by the holistic principle that the presence of one single low-cost component does not ensure that the same property holds for the whole project.

In Table 1, we give cost details for a pilot deployment according to the prototype design: we envision five WiFi Access Points (AP), 50 sensors, and three different applications. Sensors have individual 6 V 2 W solar panels, a step-down board recharging a 3.7 V 4.2 Ah Li-Ion battery, with an estimated autonomy of 20 days. Prices are found on eBay, considering bulk sales.

Table 1. Price details.

Resource	# Pieces	Cost	Unit
Arduino Mini-Pro	50	2	dollars
ESP-01	50	1	dollars
Printed circuit board	50	2	dollars
Solar panel	50	5	dollars
Battery	50	2	dollars
WiFi AP	5	50	dollars
Total hardware	-	**850**	dollars
Library	1	200	lines
Applications	3	200	lines
Total software	-	**800**	lines
ThingSpeak	1	750	dollars/year
3G subscriptions	5	24	dollars/year
Total services	-	**870**	dollars/year

The most expensive resource is the ThingSpeak subscription, which may be regarded as not strictly needed, since the operation of the pilot system fits the ThingSpeak free plan, which is reserved to non-profit experiments, as a matter of fact, with the intentions of the service provider.

We have chosen WiFi for the networking infrastructure, disregarding the highly dynamic family of LPWAN technologies, the promising LoRaWan included. To have an idea of its cost, we consider that a single gateway is sufficient for the whole smart village deployment, replacing the WiFi APs with a single LoRa gateway and reducing also the number of 3G subscriptions. Despite that, the cost for hardware equipment (considering 200 dollars for a concentrator and 15 dollars for each sensor) rises from $850 to $1400 (+65%). However, stronger reasons that make WiFi preferable today, as illustrated in the Introduction are: popularity, shareability and stability. Since such aspects are going to change in the near future, in favor of the upcoming LPWan technologies, a wise administration should prepare the ground for the change.

The sum of $2500 is comparable to a limited street maintenance task: an affordable investment even for a rural community. It may be acceptable also in developing countries, in case it helps to satisfy primary needs. The community incurs yearly expenses to sustain the system, which amount to $1000; they consist of the ThingSpeak subscription, and of the telephone company charges.

Project costs scale less than linearly with system size. The only component with linear scale-up is related to hardware: sensors and the infrastructure. The ThingSpeak fee remains unchanged until project size scales up 10 times, i.e., from 50 to 500 components, while telephone charges increase linearly. Software remains unchanged, although further revisions are needed. Yearly costs grow less than linearly, until the 500 components threshold is reached.

Under such assumption, if the project scales up ten times after the initial pilot phase, without adding new applications, we have the costs outlined in Table 2. In the scenario of an exponential growth, a successive jump, from $\times 10$ to $\times 100$, incurs a nearly linear growth of yearly expenses.

Table 2. Scale-up with scale factors $\times 1$ (pilot), $\times 10$ and $\times 100$.

Kind	pilot ($\times$1)	$\times$10	$\times$100	Unit
Hardware	850	8500 ($\times$10)	85,000 ($\times$100)	dollars
Software	800	800 ($\times$1)	800 ($\times$1)	lines
Services	870	1950 ($\times$2)	13,500 ($\times$15)	dollars/year

The sensor design presented in this paper has been successfully applied to the development of a sound-monitor for traffic analysis [36]: an amplified microphone analyses street noise using a Fourier Transform, and detects traffic jam, or passing vehicles. The results are uploaded to a ThingSpeak channel, where data can be used for statistics or real-time alerts.

There are limits in the kind of applications that are compatible with our benchmark, and they should be understood from the beginning. For instance, air pollution sensors are expensive, the management of short events is power consuming, and video processing is precluded. However, the door is open to test the ground with small scale, sustainable deployments.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Gascó, M. What Makes a City Smart? Lessons from Barcelona. In Proceedings of the 2016 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, USA, 5–8 January 2016; pp. 2983–2989. [CrossRef]
2. Van Winden, W.; van den Buuse, D. Smart City Pilot Projects: Exploring the Dimensions and Conditions of Scaling Up. *J. Urban Technol.* **2017**. [CrossRef]
3. Hogan, P.; Cretu, C.; Bulc, V. *EU Action for SMART VILLAGES*; Technical Report; European Commission: Brussels, Belgium, 2017.

4. European Union. CORK 2.0 Declaration—"A better life in rural areas". In *Cork 2.0: European Conference on Rural Development*; European Union: Brussels, Belgium, 2016. [CrossRef]
5. United States Environmental Protection Agency. *Smart Growth in Small Towns and Rural Communities*; United States Environmental Protection Agency: Chicago, IL, USA, 2002.
6. Tragos, E.Z.; Angelakis, V.; Fragkiadakis, A.; Gundlegard, D.; Nechifor, C.S.; Oikonomou, G.; Pöhls, H.C.; Gavras, A. Enabling reliable and secure IoT-based smart city applications. In Proceedings of the 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS), Budapest, Hungary, 24–28 March 2014; pp. 111–116. [CrossRef]
7. Mekki, K.; Bajic, E.; Chaxel, F.; Meyer, F. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express* **2018**. [CrossRef]
8. Adelantado, F.; Vilajosana, X.; Tuset-Peiro, P.; Martinez, B.; Melia-Segui, J.; Watteyne, T. Understanding the Limits of LoRaWAN. *IEEE Commun. Mag.* **2017**, *55*, 34–40. [CrossRef]
9. Raza, U.; Kulkarni, P.; Sooriyabandara, M. Low Power Wide Area Networks: An Overview. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 855–873. [CrossRef]
10. Balid, W.; Refai, H.H. On the development of self-powered IoT sensor for real-time traffic monitoring in smart cities. In Proceedings of the 2017 IEEE SENSORS, Glasgow, UK, 29 October–1 November 2017; pp. 1–3. [CrossRef]
11. Rizwan, P.; Suresh, K.; Babu, M.R. Real-time smart traffic management system for smart cities by using Internet of Things and Big Data. In Proceedings of the 2016 International Conference on Emerging Technological Trends (ICETT), Kollam, India, 21–22 October 2016; pp. 1–7. [CrossRef]
12. Shi, J.; Jin, L.; Li, J.; Fang, Z. A smart parking system based on NB-IoT and third-party payment platform. In Proceedings of the 2017 17th International Symposium on Communications and Information Technologies (ISCIT), Cairns, Australia, 25–27 September 2017; pp. 1–5. [CrossRef]
13. El-Tawab, S.; Oram, R.; Garcia, M.; Johns, C.; Park, B.B. Data analysis of transit systems using low-cost IoT technology. In Proceedings of the 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Big Island, HI, USA, 13–17 March 2017; pp. 497–502. [CrossRef]
14. Spachos, P.; Song, L.; Gregori, S. Power consumption of prototyping boards for smart room temperature monitoring. In Proceedings of the 2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Lund, Sweden, 19–21 June 2017; pp. 1–6. [CrossRef]
15. Desai, N.S.; Alex, J.S.R. IoT based air pollution monitoring and predictor system on Beagle Bone Black. In Proceedings of the 2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2), Chennai, India, 23–25 March 2017; pp. 367–370. [CrossRef]
16. Velásquez, P.; Vásquez, L.; Correa, C.; Rivera, D. A low-cost IoT based environmental monitoring system. A citizen approach to pollution awareness. In Proceedings of the 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), Pucon, Chile, 18–20 October 2017; pp. 1–6. [CrossRef]
17. Verma, P.; Kumar, A.; Rathod, N.; Jain, P.; Mallikarjun, S.; Subramanian, R.; Amrutur, B.; Kumar, M.S.M.; Sundaresan, R. Towards an IoT based water management system for a campus. In Proceedings of the 2015 IEEE First International Smart Cities Conference (ISC2), Guadalajara, Mexico, 25–28 October 2015; pp. 1–6. [CrossRef]
18. Suresh, M.; Muthukumar, U.; Chandapillai, J. A novel smart water-meter based on IoT and smartphone app for city distribution management. In Proceedings of the 2017 IEEE Region 10 Symposium (TENSYMP), Cochin, India, 14–16 July 2017; pp. 1–5. [CrossRef]
19. Aboelmaged, M.; Abdelghani, Y.; Ghany, M.A.A.E. Wireless IoT based metering system for energy efficient smart cites. In Proceedings of the 2017 29th International Conference on Microelectronics (ICM), Beirut, Lebanon, 10–13 December 2017; pp. 1–4. [CrossRef]
20. Singh, J.; Pasquier, T.; Bacon, J.; Ko, H.; Eyers, D. Twenty Security Considerations for Cloud-Supported Internet of Things. *IEEE Internet Things J.* **2016**, *3*, 269–284. [CrossRef]
21. Kamlaesan, B.; Kumar, K.A.; David, S.A. Analysis of transformer faults using IoT. In Proceedings of the 2017 IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), Chennai, India, 2–4 August 2017; pp. 239–241. [CrossRef]

22. Billy, L.P.L.; Wijerathne, N.; Ng, B.K.K.; Yuen, C. Sensor Fusion for Public Space Utilization Monitoring in a Smart City. *IEEE Internet Things J.* **2017**, *5*, 473–481. [CrossRef]
23. Arduino Mini Pro Schematics. Available online: https://www.arduino.cc/en/uploads/Main/Arduino-Pro-Mini-schematic.pdf (accessed on 30 April 2018).
24. Arduino UNO Schematics. Available online: https://www.arduino.cc/en/uploads/Main/Arduino-uno-schematic.pdf (accessed on 7 May 2018).
25. STMicroelectronics. STM32 32-Bit ARM Cortex MCUs. Available online: http://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html (accessed on 29 March 2018).
26. Portaluri, B. Arduino WiFi Library for ESP8266 Modules. Available online: https://github.com/bportaluri/WiFiEsp (accessed on 29 March 2018).
27. Kodali, R.K.; Mahesh, K.S. A low cost implementation of MQTT using ESP8266. In Proceedings of the 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Greater Noida, India, 14–17 December 2016; pp. 404–408. [CrossRef]
28. Baccarelli, E.; Naranjo, P.G.V.; Scarpiniti, M.; Shojafar, M.; Abawajy, J.H. Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study. *IEEE Access* **2017**, *5*, 9882–9910. [CrossRef]
29. Ciuffoletti, A. Design and implementation of a low-cost modular sensor. In Proceedings of the 13th IEEE International conference on Wireless and Mobile Computing (WiMob), Rome, Italy, 9–11 October 2017.
30. Ciuffoletti, A. A Low-Cost WiFi Sensor/actuator. 2018. Available online: http://fritzing.org/projects/a-low-cost-wifi-sensoractuator (accessed on 30 April 2018).
31. Ciuffoletti, A. A Wrapper for the AT Interface of the ESP8266. 2017. Available online: https://bitbucket.org/augusto_ciuffoletti/atlib (accessed on 30 April 2018).
32. Ciuffoletti, A. Implementation of a Benchmark for a Low-Cost Sensor/Actuator. 2018. Available online: https://bitbucket.org/augusto_ciuffoletti/benchmark_l55 (accessed on 30 April 2018).
33. Ciuffoletti, A. Sharing a common time reference in a heterogeneous distributed system. In Proceedings of the 7th EUROMICRO Workshop on Parallel and Distributed Processing, Funchal, Portugal, 3–5 February 1999; pp. 359–366.
34. Naranjo, P.G.V.; Pooranian, Z.; Shojafar, M.; Conti, M.; Buyya, R. FOCAN: A Fog-supported Smart City Network Architecture for Management of Applications in the Internet of Everything Environments. *arXiv* **2017**, arXiv:1710.01801.
35. Gammon, N. Power Saving Techniques for Microprocessors. 2015. Available online: http://www.gammon.com.au/power (accessed on 29 March 2018).
36. De Filicaia, J. Un prototipo IoT: Dall'ideazione alla Promozione. Master Thesis, Università di Pisa, Pisa, Italy, 2018. (In Italian) [CrossRef]