Sensmart: Sensor Data Market for the Internet of Things

Ricardo Miranda
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Portugal
ricardo.s.miranda@tecnico.ulisboa.p

Miguel L. Pardal
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Portugal
miguel.pardal@tecnico.ulisboa.pt

António Grilo
INESC-ID, INOV, Instituto Superior Técnico, Universidade de Lisboa
Portugal
antonio.grilo@inesc-id.pt

ABSTRACT

Currently, there are millions of sensors connected to the Internet. These sensors gather various types of data, from temperature, humidity, sound and image, to location or biometrics, to name a few. These kinds of data can be very relevant for scientific or business purposes. However, there is no online platform or marketplace where it can be easily exchanged. In this work we design and implement *Sensmart*, a solution through which it is possible to purchase and sell sensor data. *Suppliers* share their devices or data and *consumers* can buy data or acquire control of a device over a period of time. Sensmart goes beyond data exchange, and provides the ability to control a sensing device, for example, a customer can position a camera or move a robot. The platform was tested and evaluated through use cases and the implemented solution allows customers to share sensor devices and the data in an effective way.

CCS CONCEPTS

• **Hardware** → *Sensor devices and platforms*; • **Information systems** → *Data access methods*; • **Computer systems organization** → *Sensor networks*;

KEYWORDS

Sensing-as-a-service, Internet of Things, Sensors, Data, Market

ACM Reference Format:
Ricardo Miranda, Miguel L. Pardal, and António Grilo. 2020. Sensmart: Sensor Data Market for the Internet of Things. In *Proceedings of ACM SAC Conference (SAC'20).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3341105.3373951

1 INTRODUCTION

The Internet of Things *"is a world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes"* [2]. In this *everything-is-connected* world, even everyday objects can be integrated into the network, e.g. a temperature sensor, a coffee machine, a car, etc. The technology that enables such connectivity can be Bluetooth, ZigBee, or Radio-Frequency IDentification (RFID). It may change but, nevertheless, the concept of interconnection remains.

The infrastructure and the popularity of the Internet of Things make new interaction models possible, such as *Sensing-as-a-Service* [7]. This model depicts a marketplace where interested consumers trade sensor data. One good scenario for this is a Smart City [4]. It is possible to measure and sell a variety of parameters and information in an urban area, from pollution to automotive traffic, luminosity in the streets, and others.

The work described in this paper was designed, implemented and tested to provide a Sensor Data Market for the Internet of Things. The solution, called *Sensmart* (SENSor data MARkeT), considers the characteristics and limitations of different sensing devices. Additionally, it facilitates integration with existing IoT platforms and makes use of communication protocols specifically designed for sensors. In the implemented system, currency transactions were not implemented, being relegated to future work.

In the next Section, we start by analysing the state-of-the-art of Internet of Things technology.

2 BACKGROUND

Devices, communication protocols and infrastructure are building blocks of Internet of Things solutions.

2.1 Sensing Devices

One of the most common sensing devices in the world is a smartphone. The number of smartphones is expected to have reached 2.6×10^9 in 2017 and pass 5×10^9 by 2020. Additionally, according to *Perera et. al* [5] there will soon be 1×10^{10} devices connected to the Internet. We will discuss devices into two broad categories: Simple devices and Rich devices.

Simple devices perform only basic tasks, such as sending data or receiving instructions. Each sensor can measure and send data to the Internet. A network of street lamps with luminosity sensors, could, in addition to their normal functionalities, send the data measured to a marketplace platform. Dozens of temperature, humidity and air pollution sensors could be scattered in a city to create a map of the city's most polluted areas, not only to the scientific community and to the energy producers, but also to the public.

Rich devices are capable of executing tasks more complex than the simple devices, and these devices will have more processing power. Controllable cameras, Unmanned Aerial Vehicles (UAVs), or robots are examples of rich devices. A video clip or pictures of a natural landscape or monument can be acquired by such devices, and made available through a marketplace.

SAC'20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3373951

2.2 Communication Protocols

The vast majority of devices in IoT are small sensors, thus having low processing power and being very energy dependent, but also they are limited devices and cannot have a full TCP/IP stack. As the devices have limited processing capability, the communication protocol cannot be too demanding on resources.

One of the most widely known and employed application protocols is the *Hypertext Transfer Protocol*. HTTP [1] is characterized by a request/response model and the communication is established over TCP/IP. However, TCP connections do not perform well in the IoT world, due to its primary characteristic, the connection. By establishing a connection, with delivery assurance method and congestion control system, it requires significant energy and processing power resources. Furthermore, HTTP headers occupy a large portion of the message, leading to low payload efficiency. Nevertheless, it is widely available and allows high interoperability between various components of a solution. There are other protocols better suited to IoT solutions.

CoAP [2] is a specialized web transfer protocol for use with constrained nodes and constrained network, such as Wireless Sensor Networks (WSNs), and also for machine-to-machine (M2M) applications. Unlike HTTP, CoAP communicates over UDP (User Datagram Protocol) that does not require a connection. CoAP provides a request/response interaction model, in the same sense as HTTP. CoAP can use gateways to communicate with HTTP agents, if necessary. It has low header overhead, allowing more data payload in the message, and the processing power requirements are fewer. However if the message size increases, the probability of message loss in UDP becomes higher than in TCP, therefore CoAP must retransmit the whole message.

MQTT [3] is a widely adopted lightweight publish/subscribe messaging protocol. It is useful for low power sensors and machine-to-machine applications. The architecture assumed by the MQTT protocol is composed of three main elements: publisher, broker and subscriber. In MQTT a message is published to a topic. Clients may subscribe various topics and every client subscribed to a topic will receive every message sent to that topic. For the broker this is not an issue, as this element does not read the messages, it only classifies the data in topics and sends the messages to the subscribers. This broker may represent a problem: every publisher must be configured with the broker's address and the broker itself can become a single-point of failure. MQTT communicates via TCP so, as with HTTP, there may be some issues regarding the performance of the overall transmission when compared to CoAP (UDP).

2.3 Cloud infrastructure

Most IoT solutions rely on a back-end infrastructure to store data and to perform the transactions required for their operation, such as purchases. There are several IoT cloud platforms available from renowned companies: Azure IoT Hub [4], IBM Watson IoT [5], and AWS IoT [6].

[1] https://tools.ietf.org/html/rfc2616
[2] https://tools.ietf.org/html/rfc7252
[3] https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php
[4] https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-what-is-azure-iot
[5] https://cloud.ibm.com/docs/services/IoT/index.html
[6] https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

The Azure IoT Hub provides a collection of services for connecting and extracting data from sensing devices. It enables bidirectional communication with the devices and the cloud, in a variety of communication protocols, directly or through a gateway. Furthermore, it exposes an API capable of sending messages to the devices, if the devices support this mechanism.

The IBM Watson IoT Platform provides a versatile IoT toolkit to connect, manage and extract data from a wide variety of sensing devices. Watson IoT Platform can communicate or interact with external applications, thus having the ability to export the data collect to Databases or Management Dashboards.

The AWS IoT communication architecture revolves around the concepts of topic and state. The *state* can be a value, a keyword or a symbol. The *topic* acts like a channel: a device publishes a message to one topic and clients can subscribe to these topics to receive the new messages sent by the devices. When a device (sensor) sends data, it updates the current state of a property. This information is stored in the *Thing Shadow*, a set of properties about a physical object. When one device is registered in the AWS IoT, a shadow is created for it. A device connects and sends/receives data to the AWS over MQTT, HTTP or WebSockets. The Gateway and the Message Broker allow secure and low-overhead communication between the devices, the cloud and other applications. By deploying a Rule, via the Rule Engine, it is possible to store or process data sent by sensors, or even filter it. Amazon developed a set of SDKs from which a group was designed to be used on micro-controllers/resource-constrained devices.

Regarding security, the presented cloud solutions offer similar schemes. IBM IoT allows for different security policies, characterized by combinations of authentication via certificate, via token, or via both. AWS IoT can generate a certificate automatically when a new device is registered in the platform. The certificate is used for device authentication which is more secure system than other schemes, such as user name and password. All platforms implement TLS connections in client (device) to server communication.

2.4 Sensing-as-a-Service systems

Perera et. al [4] discuss a detailed architecture of a sensing-as-a-service system, and more specifically a possible structure for its implementation in a Smart City. The authors describe the various layers of the model and categorize various classes of information, sensors/devices and their owners. It gives examples of some application opportunities within a Smart City and also describes three *imaginary* operation scenarios. In the end the authors present several advantages and benefits of the sensing-as-a-service model.

Chang et. al [1] focus in a framework for mobile sensing-as-a-service. The authors discuss and classify types of sensing services, sensing activities and devices. The architecture considers the providers, the clients and the cloud platform service. The latter is developed to connect all the other elements. The framework relies on the cloud service as a proxy to deliver the data from the supplier to the clients.

Montserrat [3] developed a software allowing a client to share his sensors with another user of the platform. The author also discusses the business model and implements a web application to manage the sensor network.

The solutions presented in [4] and [1] provide a foundation for future works. The Cloud in [1] is a crucial element in the architecture of a marketplace platform and the analysis [4] of sensors and information is determinant to properly develop a storage system. Our solution has similar goals to the work [3], but presents a more detailed analysis of the communication protocols, the architecture is different and the solution has more functionalities.

3 SENSMART

We present our proposal, *Sensmart* (SENSor data MARkeT), starting with the architecture, and then focusing on the Marketplace operation with simple and rich sensor products.

3.1 Architecture

The architecture of the proposed solution is presented in Figure 1 and is composed of three modules: Device, Broker and Marketplace. Each one has a defined Application Program Interface (API).

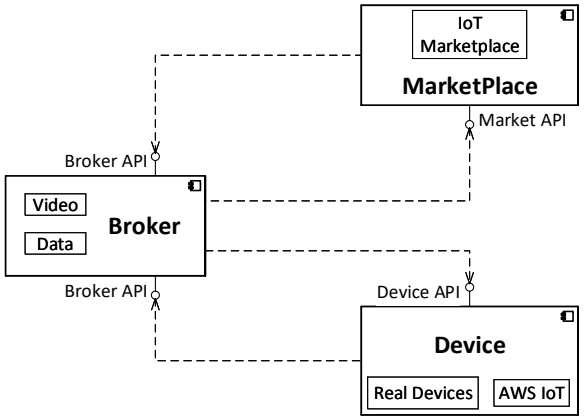

Figure 1: General architecture of the Sensmart system.

3.1.1 Device Module. When designing and developing an architecture for a given application, the variety of devices may present an impediment or a difficult problem to solve due, for example, to the integration of different hardware or communication protocols. Not all devices can communicate in the same manner, since they may be designed to use different radio frequencies and communication protocols. Additionally, the message payload format may be different, from device to device. Sometimes bitwise operations are required to translate the payload into intelligible information. Gateways may be needed to hide the heterogeneity of devices from the information systems that use them. Generically, a *gateway* can be used with software or hardware implementation that enables the communication between two sides.

As discussed in Section 2.3, the AWS IoT's Shadow Thing is a virtual representation of a real sensing device. The integration of AWS IoT into the Device module provides a simpler and wider communication interface for the Device module to export. A sensing device publishes its data to its topic via the MQTT protocol, for example. The Broker module is able to interact with a topic by means of AWS IoT's interface. Also, the sensor's owner can send control messages to the sensor through the same method, i.e., the owner can send a message to reboot the sensor via the Shadow Thing topic. Therefore, the Device module is composed of actual devices, gateways and the AWS IoT platform.

3.1.2 Broker Module. The Broker module is the link between the marketplace and every sensor. This module is able to communicate with both Device module and Marketplace module. In greater detail, the Broker module is implemented through a Data/Video Broker and Data/Video Handler. The first is responsible for receiving and managing new requests originated from the Marketplace, while the latter, establishes and controls the communication path between the sensing device and the buyer/seller. Both components have different implementations for each type of sensor. Nevertheless, the Broker always communicates with the Marketplace module via a RESTful HTTP API. The Broker defines a set of URLs allowing for the Marketplace or the Device modules to send requests, i.e., purchase orders, data or control commands.

3.1.3 Marketplace Module. The marketplace is the intermediary between the buyer (who wants to gather data from sensors) and the seller (who has data or sensor control to sell). This module aims to create an environment, where, for example, a person can place a temperature sensor on the street where he/she lives and sell the measurement data to a meteorological service.

Any entity will be able to search through the list of available devices, select one or more, and send a request. This request can be to acquire data (already collected or a future reading), or to control a sensor. The owner of a sensor can designate a rate or price for a single data measurement or a collection of measurements. This *entity* is a representation of a person or company. An entity can own a sensor or device and buy or sell data. For example, subject A buys data from one sensor that belongs to subject B for a cost of X. The platform will transfer an amount to A and will send an instruction to the Broker module to transfer the purchased data to subject B. Each device is associated with one entity. Nevertheless, the latter can *have* multiple devices or none. To implement this relation, a virtual replica of a device is created. This virtual device is a Shadow Thing from AWS IoT platform. When a client purchases real-time data from a sensing device, the data is published to the sensor's topic. In order to receive the data, the client *listens* to the sensor's topic, via his/her virtual device. The whole architecture is developed around the role of the virtual device. This entity can identify a real sensor, but it is also possible to use it just as a virtual device. This issue is addressed in Section 3.2.1 in greater detail.

3.2 Marketplace Operation

The marketplace is the core element of Sensmart. Through this platform, consumers and suppliers are able to exchange data and control of sensing devices for currency. The marketplace was implemented in Java [7] and its interaction involve the following five elements: *Customers*, persons or entities that purchase and sell data; *Sensing devices*, which collect data or to be controlled; *Catalogs*, which store and present sensing devices to the customers; *Stock*, corresponding to the inventory of each sensing device; *Broker*, which is the bridge between the sensing device and the marketplace.

[7] https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html

If one of these elements is not present, the marketplace will not succeed. Without sensing devices, there is nothing to offer to the customers, no data to be collected and no device to be controlled. If it is not possible to organize and store the devices in a presentable and searchable way, the customers will never find the device or data they wish. If a stock/inventory is not implemented, although it is possible to find sensing devices (through the catalog), when a customer attempts to purchase data, the device might be offline or not available at that moment. Thus, the client would pay for something that he would never receive. Even worse, the device, a controllable device, could already be in use by another customer, resulting in two customers trying to control the same device at the same time. Without a communication path between the marketplace and the sensing devices, the customer will never receive any data measurements or will never be able to control a device. Lastly, if there are no customers, the marketplace will fail, for obvious reasons.

The marketplace interacts with each of the previously presented elements. Firstly, it can create Customers, Devices and Products. Next, the Marketplace can manage the list of Customers and the list of Devices, namely the Catalogs. It is possible to add, retrieve or delete elements or find a specific sensing device or client. Regarding a Stock, the Marketplace can add and retrieve a Product, verify if it is available to purchase or mark it as unavailable. At this point, two customers cannot exchange data for currency yet. In order to exchange data or control of a sensing device, the Marketplace interacts with the broker. Sale specification or sale spec, is the designation of a set of data crucial to perform any exchange in the marketplace. This information is in JSON format and is sent with the HTTP request to the Broker or to the database. Considering the item being purchased, the sale spec may contain the seller's name and topic, the buyer's topic (associated with the AWS IoT) and the time interval. For a different type of item, it may contain the IP address of the video stream and of the control module, amongst other information. The sale spec is critical. Without it, the Broker will not be able to connect the buyer and the data/sensor.

3.2.1 Simple Sensor Products. The broker's module implementation for sensing devices is composed of: AWS IoT platform, the foundation of the communication infrastructure; Data Broker, the entry point of communication; Data Handler (simple mode), responsible for managing the customer-device connection in *simple* data transactions; Data Handler (long mode), responsible for managing the customer-device connection in *long* data transactions.

The AWS IoT platform was selected to be the infrastructure behind the implementation for the simple sensors. Through AWS IoT platform, each simple sensor owner has a Shadow Thing, a virtual representation of the real sensor. However, a Thing can also be a logical entity, i.e., it may not represent a real device. This characteristic is particularly useful to represent a buyer, as it will be explained below. Every simple sensor is connected to the AWS IoT platform via its Thing and publishes data to a topic, designated as topic S (Seller). Consider the following: a customer/buyer purchases all data measurements from a sensor for a period of time; All data messages from the sensor are published to topic S and, since, the customer cannot communicate directly with the sensor, the customer does not have access to topic S. As presented above, a Thing can also be a logical entity. By associating a Thing to the customer and creating a topic, topic B (Buyer), it is now possible for the buyer to receive all the data measurements published in topic S. The Broker, namely the Data Handler, will subscribe to topic S and copy the messages to topic B.

Device: In order to illustrate a simple sensor, a temperature sensor was implemented. The CPU temperature sensor of a Raspberry Pi was used for this purpose. The Raspberry Pi is connected to the Internet in order to be able to send the data it measures and to receive instructions. The described setup aims to simulate a sensing device composed of a temperature sensor and a GSM/WiFi module. As pointed above, the AWS's library that was implemented communicates via the MQTT protocol. The device's communication module or gateway was also implemented by means of the same AWS library, thus ensuring compatibility. The software implemented in the device executes two tasks recurrently: temperature measurement and data message publication. Also, right before publishing a new message, the device verifies if there is a new message directed to itself, such as a restart command. If there is, it processes that message first.

Data Broker: The Data Broker is an HTTP server designed to process every data transaction sent from the marketplace module. The data broker isolates the data provider from the data consumer. This is done to ensure that the consumer cannot bypass the marketplace and later access the device directly. For this reason, the device is always *indirectly* contacted.

Two modes of data transaction were defined, namely *simple* and *long*. When a customer/owner wishes to send a command or control message to a sensor or requests a data measurement at a specific time-instant, these situations are considered to be *simple* data transactions. On the other hand, a *long* data transaction can be described as a request that occurs for a time period. For example, a customer wishes to collect all measurements from a sensor starting at 13h00 and ending at 16h00. Hence, is it important to process each transaction type separately. An HTTP request containing the salespec is sent to the data broker. The request is validated and upon success the broker will create a sub-process of a Data Handler. Finally, the Data Broker will resume and wait for a new HTTP request.

Data Handler (simple mode): The Data Handler for *simple* transactions was designed to send a control message to a device or to request a data measurement and deliver the data to the customer. The broker module should handle every data transaction, ensuring that the costumer can never interact directly with a device. The sale specification of each purchase/request is enforced, and the device is only required to publish data measurements. Thus, the Data Handler is the intermediary between the sensor and the customer. Consider the following scenario: a customer requests a data measurement at 16h00. The sensor publishes data or messages in a topic, designated as topic S. The Data Handler receives the sale specification from the Data Broker. Afterwards, it subscribes to topic S and sends the data measurement request or the control message to the sensor via the sensor's shadow topic, as previously discussed. Upon receiving a message from topic S, the Data Handler

verifies if the time of reception of the message satisfies a predetermined deviation margin. If successful, the message is published to the buyer's topic, topic B. Otherwise, when the instant of reception surpasses the upper limit, an error message is published to topic B. However, if it does not reach the lower limit, the Data Handler remains listening to topic S. Finally, after publishing to topic B the Handler terminates itself.

Data Handler (long mode): This Data Handler was designed to subscribe the seller's sensor topic and publish any received message to the buyer's topic and acting as intermediary. Consider the following example: a customer wishes to receive every data measurement between 16h00 and 18h00. When a customer purchases data measurements during a period of time, the Data Handler subscribes to topic S. Upon receiving a new message, the Data Handler will publish that message to the buyer's topic, topic B. This interaction terminates when the current time surpasses the end time of the purchase. At this moment, the Data Handler shuts down and the buyer will no longer receive messages, in topic B, from sensor.

3.2.2 Rich Sensor Products. As shown in Section 3.1.2 the marketplace module does not communicate directly with the device module. The agent responsible to connect those modules is the broker. Moreover, after a successful purchase of a time-slot on a rich sensor, the customer will communicate with the sensor via the broker module. With this module, the buyer can control the device or receive any type of data stream, may it be video or sound. In the case of video, the broker module is composed of: Video Broker, the entry point of communication; Handler, responsible for managing the customer-device connection; IP/TCP Tunnel, the communication channel; Web page, responsible for presenting the video stream and control mechanism.

Device: The rich sensor implemented is an IP controllable web camera. This device can create a video stream and associate it to an IP and a port by creating a server. As it is controllable, the control module has an IP and port also. The separation between video and control is essential. Ensuring a division between the two prevents a customer, which only purchased the visualisation and not the control, to use the video IP/TCP connection to attempt to control the camera. The device's control module is a program developed in C programming language. This program manages a TCP socket to receive control messages from the Handler's server, translate them and move the camera.

Video Broker: The Video Broker is a server designed to handle each HTTP request sent from the marketplace. After determining if the request itself is valid, it will process the salespec sent by the marketplace. Once processing is finished (with success), the broker will create a file to store all the information from the specification, create an instance of Handler and send a reply to the marketplace. The reply contains the port associated to the HTTP server of the Handler previously created.

Video Handler: A Handler will not only process each control request sent by the customer, but also manage the IP/TCP connections for the video stream and to control the device. By opening the web page, the customer activates a mechanism that sends an HTTP request to the Handler's server. This request triggers the creation of a connection (IP/TCP tunnel) between the customer's address and the video stream address. This connection acts as a link. From the customer's perspective, the address of the video stream is the Handler's server address, thus ensuring that the video stream address is never shown. Once a customer sends a control request, via one of the buttons from the web page, the server will authenticate the origin of the request. Upon a positive authentication, the request is parsed and sent to the device's control module. When the current time reaches the end of the time window each Handler's instance terminates itself. Hence, each customer will no longer be able to view the video stream or control the device.

4 RESULTS AND EVALUATION

The results presented in this section comprise three scenarios: single data measurements, long data measurements, and video and control. The section finishes with an overall discussion of the results.

4.1 Single Data Measurements

This experiment consists of scheduling three hundred measurements to the same sensor and receiving the respective data. In greater detail, the assessment is divided in three series of one hundred measurements, respectively. For each series, the Marketplace is set up with a pool of sensors, buyers and sellers. One buyer and one sensor are selected and each data request is originated from the same buyer to the same sensor. In the Marketplace, each request is generated with a one-second interval. However, from the sensor's perspective, two consecutive data measurements are separated by a sixteen second interval, i.e., the sensor measures the temperature every sixteen seconds.

To better evaluate the performance of the Broker and the Handlers, the requests are created with one second interval between them. Since every measurement is set to occur in the future, the Handler will be inactive until such time is reached.

Figure 2 presents the time difference between the reception time and the scheduled time of each measurement. These values have a high accuracy, since the marketplace (which originates the scheduling timestamp) and the client (which originates the reception timestamp) share the same clock (both programs were running on the same machine). The depicted data were collected throughout the third series of one hundred measurements. In this series, the mean value for the time difference is 1.23 seconds, represented by a line in the graph. While analysing the graph, it is possible to observe that a considerable number of samples are in the interval from 1 to 1.5 seconds. Moreover, only six samples are above the 2 second mark, being that the maximum in the series is 3.44 seconds.

To evaluate the performance of the proposed solution, it is crucial to assess the time required to process each purchase order or each message replication. Figure 3 presents the proportion of processing time to the total time, identifying its components using different colors. The Marketplace takes more than 50%, the Handler more than forty percent and the Broker consumes no more than one percent. These values are in accordance with what was stated in Section 3.2.1.

Table 1 presents the overall statistical results calculated from the data collected throughout this experiment.

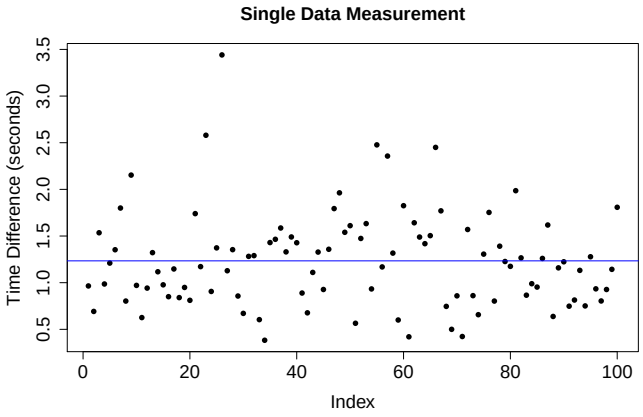

Figure 2: Temporal discrepancy between reception and scheduled times.

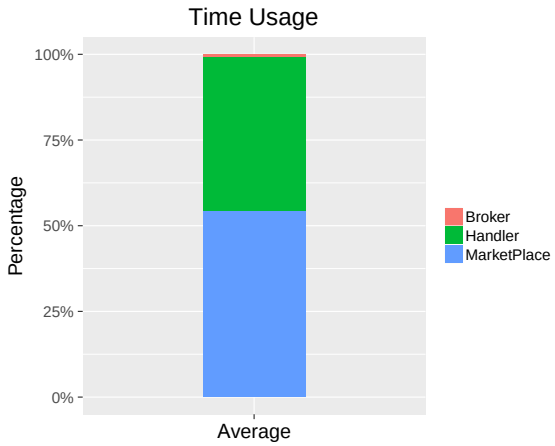

Figure 3: Average time usage for each participant regarding a data transaction.

Considering that the Marketplace only finishes processing when it receives a reply from the Broker and the Client only receives a message after it is published by the Handler, one would expect that a high processing time in the Broker or in the Handler could interfere with the processing time of the Marketplace or delay a message to the Client. Nevertheless, this hypothesis is not verified. Analysing the collected data, a higher time difference is not due to a higher processing time in the Handler. The variation in the time difference, for instance, may be caused by the processing inside the AWS IoT platform itself.

In Table 1, the mean time difference is 1.35 seconds. Picturing a real scenario where a client schedules a measurement from a sensor, this value indicates that the service is feasible. From a set of three hundred scheduled measurements, only one measurement was not received by the client, which demonstrates a high success rate.

4.2 Long Data Measurements

This experiment is designed to analyse the performance of the implemented solution when it executes one of the main operations.

Table 1: Single Data Results: Processing time for each actor and overall time difference (seconds).

	Market	Broker	Handler	Time Diff.
Mean	1.05E-01	1.66E-03	8.68E-02	01.35
Standard Deviation	2.09E-02	5.20E-04	4.72E-03	00.63
95% Confidence Interval	2.37E-03	5.88E-05	5.34E-04	00.07
Maximum	3.38E-01	9.00E-03	1.00E-01	03.60
Minimum	9.87E-02	1.37E-03	8.00E-02	00.32

The procedure consists in purchasing real-time sensor data during a time interval. In greater detail, the assessment is divided in three series of forty-five minutes, respectively. For each series, the Marketplace is set up with a pool of sensors, buyers and sellers. Five buyers and one sensor are selected and each buyer generates a long data measurement request to the same sensor. Those requests are created with a one-second interval. The sensor was configured to measure and publish data to its topic every thirty seconds, even if there is no one listening. Considering the sensor does not have the ability to know if there are any subscribers to its topic, it will publish data continuously. Each buyer purchases forty-five minutes of subscription to the sensor's topic. Bearing in mind the thirty-second halt at the sensor, it is expected that each buyer obtains ninety measurements. To approximate this experiment to a potential real scenario, instead of having only one buyer listening to the sensor's topic, all five buyers are listening to the topic simultaneously.

Throughout the experiment, each client, stores all received messages and generates a timestamp for the instant of reception of those messages. As stated on Section 3.2.1, one message contains the measured temperature and the timestamp of measurement. The sensor is a Raspberry Pi, wherein the time and date are configured with the default specifications. All five Clients were running on the same machine, thus having the exact same clock. Bearing in mind that clients and sensor were on different machines, therefore not being completely synchronized with each other, a 1 second maximum desynchronization is considered. This 1 second is the maximum desynchronization between the client's clock and the sensor's clock. This value was obtained through visual inspection and experimentation. Even though, it is not possible to guarantee total synchronization, Figure 4 presents the time difference between instant of reception (mean) and the instant of measurement.

These samples were collected during one of the three series from this experiment. It is possible to observe that, in only two occasions, the delay between the measurement and reception is greater than 0.6 seconds, wherein the maximum time difference is 0.93 seconds. Also, more than 60 percent of the samples are below the mean value. Even considering a desynchronization of 1 second (the sensor's clock is 1 second ahead of the actual time), the mean time difference, in that situation, is 1.47 seconds. In a real scenario, the client intends to receive every measurement taken by the sensor in real-time, i.e., with a delay small enough to satisfy most applications. Table 2 presents the overall statistical results calculated from the data collected throughout this experiment.

As discussed in Section 2, the results do not support a causality relationship between a higher processing time and subsequent

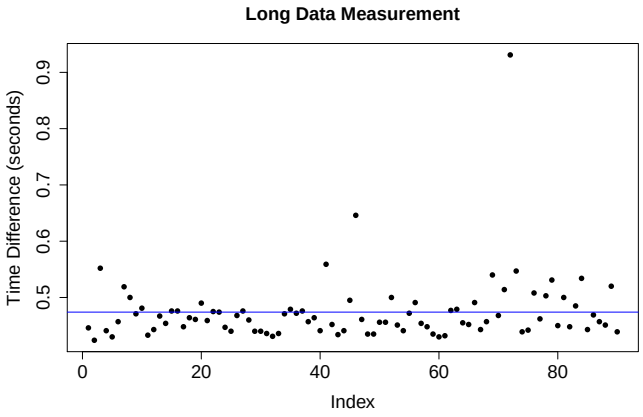

Figure 4: Temporal discrepancy between reception and scheduled times.

Table 2: Long Data Results: Processing time for each actor and overall time difference (seconds).

	Market	Broker	Handler	Time Diff.
Mean	1.06E-01	1.44E-03	8.67E-05	00.51
Standard Deviation	7.36E-03	7.10E-05	1.01E-05	00.10
95% Confidence Interval	3.73E-03	3.59E-05	5.41E-07	00.01
Maximum	1.30E-01	1.63E-03	1.68E-04	01.49
Minimum	9.98E-02	1.37E-03	7.20E-05	00.42

delays when receiving a message. In this experiment, the mean processing time of the Marketplace and the Broker are similar to the values from Table 1. However, the Handler has a smaller processing time when compared to the previous experiment. In this experiment, the Handler subscribes the sensor's topic and publishes to the buyer's topic. Furthermore, the Handler must also alert the sensor. This additional task explains the extra processing time.

4.3 Video and Control

This experiment is designed to analyse the performance of the implemented solution when it executes one of the key operations. The procedure consists in watching a video stream from a video camera whilst having the ability to control the camera in real-time. In opposition with the previous experiments, this assessment is aimed at more complex sensing devices, e.g., devices capable of creating a video feed and accepting control requests. This assessment is divided in three series of one minute, respectively. In each series, the Marketplace is set up with a pool of sensors, buyers and sellers. One buyer and one video camera are selected to perform this test. The buyer generates one video and control request directed to the selected camera. This request contains the start time and end time, among other specifications. When the time limit is reached, the video stream ends and the buttons no longer control the video camera.

To evaluate the performance, several timestamps, processing times and messages were collected throughout this experiment. In order to increase the accuracy of this experiment, the agent that generates the message (to control the camera) and sends it to the Handler is replaced by a helper application called Postman[8]. This application issues the control order, an HTTP request, to the Handler. Postman runs a set of four HTTP requests seventeen times. Two consecutive requests are separated by five-hundred milliseconds. This configuration results in a total of sixty-eight instructions during the one-minute test period.

For every control order issued, Postman determined the response time, i.e. the time required to receive the HTTP reply from the Handler, considered to be the total time. The Handler calculated the time necessary to create and communicate the control message to the control module of the video camera, designated TCP, and estimated the total time required to process the request, designated Processing. This time includes a simple authentication verification, the TCP time and the HTTP reply to the client (Postman). Figure 5 presents the total time and the fractions due to each component. Approximately sixty percent of the total time is due to the Communication component. This element is considered to be the time spent outside the Handler and Postman. Analysing the Handler, the TCP component is responsible for nearly ninety-eight percent.

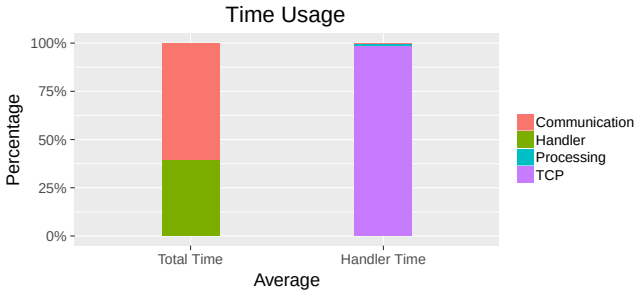

Figure 5: Average time usage for each component regarding a Video and Control transaction.

Table 3 presents the overall statistical results calculated from the data collected throughout this experiment. The column Aggregate represents the total time registered in the Handler, i.e., the time required by the TCP communication plus the processing time. The maximum values registered for both Postman and Handler (Aggregate) occur in the first control order. According to [6] the one-way latency for interactive video applications should be no more than 150 milliseconds. The measured latency corresponds to the processing and reply of the control request, thus a two-way latency with a mean value of approximately 200 milliseconds. This result is acceptable, considering that the two-way latency should be less that than 300 milliseconds. The mean value required for the Handler to fully process a request in approximately 80 milliseconds, thus having some impact in the delay between instructing the camera to move and observing that movement in the video stream.

4.4 Discussion

While developing and executing the two experiments for the Simple Device, the sensor sent more than one thousand messages through

[8] https://www.getpostman.com

Table 3: Video and Control Results: time spent in the TCP connection and overall two-way latency (seconds).

	Aggregate	TCP	Postman
Mean	5.73E-02	5.65E-02	2.01E-01
Standard Deviation	3.44E-03	1.61E-03	1.28E-02
95% Confidence Interval	4.72E-04	2.21E-04	1.76E-03
Maximum	8.23E-02	6.48E-02	2.84E-01
Minimum	5.39E-02	5.275E-02	1.93E-01

the AWS IoT. Only five messages were not received by the client, leading to a 0.005 fail ratio. Overall, the implemented solution proved to be: robust (when the customer attempted to open the video URL in another computer or browser, the Video Handler detected this abnormal behaviour and cancelled the request); effective (considering a 0.005 fail ratio in data transaction tests); fast (a low-delay was visually observed during the video and control experiments, as well as during data transactions). The mean delay between measuring and receiving is 1.54 seconds for the Single Measurements and 0.51 seconds (or 1.51 seconds, considering a 1 second desynchronization) for the Long Data Measurements, which are suitable values. In the final experiment (Rich Device), both the video stream and the control module proved to be responsive and accurate. The results were obtained using a personal computer to run the Marketplace, a regional virtual machine from Amazon Web Services to host the Brokers and Amazon Web Services IoT platform to support the Device module. Configuring dedicated machines to host the Marketplace and the Brokers could improve the results. Optimizing AWS IoT platform to the specific requirements of the Sensor Data Market may decrease the delay and the setup time necessary to establish the connections to the platform. Nevertheless, the delay between receiving a data measurement and the instant of the measurement itself may not be an issue for some applications or scenarios. For instance, a network of constrained sensing devices (sensors with low processing power and limited connectivity) may collect data every two hours. The data is stored locally in the devices and uploaded once a day to a gateway. After receiving the data measurements of all sensors, the gateway publishes it to the AWS IoT platform. In this scenario the delay should not be an evaluation parameter.

5 CONCLUSION

We proposed *Sensmart*, a Sensor Data Market for the Internet of Things. It is an intermediary between data suppliers and data consumers, and sensors are made available to any customer, in a service-like structure. The Amazon Web Services Internet of Things platform was integrated in the proposed solution, due to its architecture and support for MQTT and HTTP protocols. The AWS IoT infrastructure regarding the MQTT protocol is the foundation of the simple device implementation. The rich device is controlled using HTTP. Two sets of tests were designed and conducted, one set for the simple device and the other for the rich device. The example simple device was a temperature sensor and the rich device was a remote-control web camera. Considering the results obtained and presented before, we conclude that the platform prototype designed and implemented in this work is a practical solution for a Sensor Data Market for the Internet of Things.

Some future work points can improve the current solution: implement more types of devices to widen the range of sensors that can be integrated in the marketplace, thus strengthening the offerings of the Sensor Data Market; design and implement a web interface to interact with the platform will enrich the current solution; support widely used database engines, to better store information of customers, accounts, transactions, and devices; integration with currency transactions, both based on cryptocurrency and regular money; finally, security and privacy concerns must be addressed, both to protect buyers and sellers, but also people potentially surveilled by these kinds of systems.

ACKNOWLEDGEMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 (INESC-ID), through project with reference PTDC/CCI-COM/31440/2017 (SureThing) and through FITEC - Programa Interface, with reference CIT "INOV - INESC Inovação - Financiamento Base".

REFERENCES

[1] C. Chang, S. N. Srirama, and M. Liyanage. 2015. A Service-Oriented Mobile Cloud Middleware Framework for Provisioning Mobile Sensing as a Service. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. 124–131. https://doi.org/10.1109/ICPADS.2015.24

[2] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. 2009. *The Internet of Things in an Enterprise Context.* Springer Berlin Heidelberg, Berlin, Heidelberg, 14–28. https://doi.org/10.1007/978-3-642-00985-3_2

[3] Eduard Montserrat and Fatos Xhafa. 2015. *Sensors as a Service in the Cloud.* Master's thesis. Universitat Politècnica de Catalunya, Spain.

[4] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2014. Sensing As a Service Model for Smart Cities Supported by Internet of Things. *Trans. Emerg. Telecommun. Technol.* 25, 1 (Jan. 2014), 81–93. https://doi.org/10.1002/ett.2704

[5] C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, and D. Georgakopoulos. 2014. Sensor Search Techniques for Sensing as a Service Architecture for the Internet of Things. *IEEE Sensors Journal* 14, 2 (Feb 2014), 406–420. https://doi.org/10.1109/JSEN.2013.2282292

[6] Tim Szigeti and Christina Hattingh. 2004. *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs.* Cisco Press.

[7] Arkady B. Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. 2013. Sensing as a Service and Big Data. *CoRR* abs/1301.0159 (2013). http://arxiv.org/abs/1301.0159