technologies

MDPI

Article

ThingsLocate: A ThingSpeak-Based Indoor Positioning Platform for Academic Research on Location-Aware Internet of Things

Luca De Nardis [1,*], Giuseppe Caso [2] and Maria Gabriella Di Benedetto [1]

[1] Department of Information Engineering, Electronics and Telecommunications, Sapienza University of Rome, Via Eudossiana 18, 00184 Rome, Italy
[2] Department of Mobile Systems and Analytics, Simula Metropolitan Center for Digital Engineering, Pilestredet 52, 0167 Oslo, Norway
* Correspondence: luca.denardis@uniroma1.it; Tel.: +39-06-4458-5479

Received: 15 June 2019; Accepted: 14 July 2019; Published: 16 July 2019

Abstract: Seamless location awareness is considered a cornerstone in the successful deployment of the Internet of Things (IoT). Support for IoT devices in indoor positioning platforms and, vice versa, availability of indoor positioning functions in IoT platforms, are however still in their early stages, posing a significant challenge in the study and research of the interaction of indoor positioning and IoT. This paper proposes a new indoor positioning platform, called ThingsLocate, that fills this gap by building upon the popular and flexible ThingSpeak cloud service for IoT, leveraging its data input and data processing capabilities and, most importantly, its native support for cloud execution of Matlab code. ThingsLocate provides a flexible, user-friendly WiFi fingerprinting indoor positioning service for IoT devices, based on Received Signal Strength Indicator (RSSI) information. The key components of ThingsLocate are introduced and described: RSSI channels used by IoT devices to provide WiFi RSSI data, an Analysis app estimating the position of the device, and a Location channel to publish such estimate. A proof-of-concept implementation of ThingsLocate is then introduced, and used to show the possibilities offered by the platform in the context of graduate studies and academic research on indoor positioning for IoT. Results of an experiment enabled by ThingsLocate with limited setup and no coding effort are presented, focusing on the impact of using different devices and different positioning algorithms on positioning accuracy.

Keywords: IoT; indoor positioning; WiFi fingerprinting

1. Introduction

Internet of Things (IoT) is rapidly becoming one of the largest markets for wireless telecommunications: although exact figures depend on the definition of IoT, analysts estimated a total value market of about 600 billion dollars back in 2015, expected to reach 700–800 billion dollars by 2023 [1,2], with over 40 billion devices connected worldwide [3]. Position information is considered a cornerstone in IoT deployment, as part of the general concept of context awareness that is expected to characterize IoT devices and networks [4]. Health care [5], transportation and fleet management [6], and smart factories [7], are all IoT application areas where the availability of position information can significantly improve security and efficiency. As a result, the problem of determining the position of IoT devices has received significant attention, in particular in indoor environments where Global Navigation Satellite Systems (GNSS) are not available. Several approaches have been proposed, relying on a single wireless technology, such as WiFi [8–10], Bluetooth [11,12], or RFID [13]. A review of how suitable different wireless technologies are as the underlying technology for indoor positioning in IoT

is provided in [14]. The combination of multiple technologies is also an appealing approach, since the IoT is expected to emerge as the combination of networks of devices equipped with a wide range of wireless interfaces: furthermore, modern smartphones are equipped with multiple wireless interfaces as well as additional sensors [15]. Many IoT devices will however support a single technology in order to limit complexity, cost, and energy consumption: as a consequence, this work will focus on indoor positioning based on a single technology, and in particular on WiFi. Please note that the platform introduced in this work can support technologies different from WiFi, as will be discussed later in the paper. WiFi was however selected since it is currently the key technology for IoT deployment for mainly two reasons: (a) it is by far the most common wireless interface currently deployed in the market, with about 7 billion active WiFi devices and more than 10 billion expected by 2019 [16], and (b) it offers a straightforward solution for access to the Internet, given the presence in almost every household or industrial setting of a WiFi infrastructure, whereas other technologies require a dedicated gateway to provide Internet access. Several approaches based on WiFi have been recently proposed to provide indoor positioning in the context of IoT. A remarkable centimeter accuracy was achieved in [10] using WiFi hardware by collecting Channel Impulse Responses (CIRs). CIRs from several different WiFi channels were collected in known locations using a frequency hopping approach, and combined to define fingerprints. These fingerprints were then compared with fingerprints collected by a target device to estimate its location. In [9] the same authors achieved similar accuracy by replacing frequency diversity with space diversity, obtained by using multiple antennas. Both approaches, although aimed at IoT applications, require advanced hardware features and signal processing capabilities, and are thus not suitable for most WiFi IoT devices. An approach arguably better suited for IoT WiFi devices is Received Signal Strength Indicator (RSSI) fingerprinting [17], [18], as also pointed out in [14]. RSSI fingerprinting typically operates in two phases, offline and online. During the offline phase RSSI measurements from WiFi Access Points (APs) are collected in a set of predefined positions, referred to as Reference Points (RPs), and stored in a database in a positioning server. In the online phase, target devices collect RSSI measurements and transfer them to the server, which estimates their positions based on the similarity between the offline vs. online RSSI data. In the broad context of wireless networks, a vast literature exists on algorithms and metrics to be used in the online phase, ranging from low-complexity *k*-Nearest Neighbor (*k*NN) algorithms operating on Euclidean distance to more complex, hierarchical algorithms adopting probabilistic metrics [19]. In the specific context of IoT, in [8] an improved version of the *k*NN fingerprinting algorithm was proposed, but no specific provision was given on the hardware and processing requirements for its deployment in IoT systems.

The stated goal of the above-mentioned works was to achieve a high positioning accuracy, following along the line of the huge research efforts dedicated to accurate indoor positioning in the past 15 years. Accuracy is however only one of the aspects that should drive the design of an indoor positioning platform for IoT, since the following IoT-specific characteristics should be taken into account:

- Ease of use and ease of deployment—IoT is expected to pervade and integrate seamlessly in both consumer and industrial applications. This goal can however only be achieved if IoT devices and systems are easy to deploy and setup, not requiring extensive calibration phases and/or complex setup procedures;
- Hardware limitations—IoT devices range from very simple devices, with a single wireless interface and minimal computing/processing power, to powerful processing stations equipped with multiple Radio Access Technologies (RATs) and accessory sensors;
- Cloud computing—IoT is inherently relying on Internet connectivity and on the availability of processing power in the cloud, allowing simple IoT devices to operate by just collecting and transferring data, without local processing.

Existing indoor positioning platforms fail however to take into account the specificity of IoT devices and applications, due to the fact that IoT platforms and indoor positioning platforms evolved so far all but independently. The lack of integration between indoor positioning and IoT is indeed

a major hurdle to the successful design and deployment of location-aware IoT applications, despite recent efforts to address it [20], as will be further discussed in Section 2. Experimental activity on indoor positioning for IoT devices currently requires in fact to learn (a) how to develop on the selected IoT platform, requiring expertise on both its hardware and software, (b) how to develop on the selected indoor positioning platform, most often using a different programming language from the IoT platform, and (c) how to put them together. This complex and time-consuming workflow creates a significant obstacle to research and development for location-aware IoT applications. Such a workflow is particularly challenging in academic environments, in which graduate students are expected to learn how to master a topic in a limited amount of time during their courses or thesis work.

This work proposes an open-source platform for indoor positioning in IoT, named ThingsLocate that addresses the above limitations and bridges the gap between IoT and indoor positioning, providing a user-friendly tool for the study, research, and deployment of indoor positioning in IoT and location-aware IoT applications. Platform characteristics are described in detail, and instructions for setup and deployment are provided. A proof-of-concept deployment is also introduced, in order to show the capabilities of ThingsLocate.

The paper is organized as follows. Section 2 analyzes previous work related to WiFi-based indoor positioning in the context of IoT. Section 3 discusses existing development environments for IoT applications, and focuses in particular on ThingSpeak, adopted as the basis for the design of ThingsLocate. Next, Section 4 introduces the ThingsLocate platform, describes its features as well as the procedures to setup, deploy, and use the platform. Section 5 presents the proof-of-concept implementation of ThingsLocate, and shows how it was used to analyze the impact of different devices and different algorithms on the positioning accuracy achievable by IoT devices. Section 6 discusses extension possibilities and limitations determined by the ThingSpeak environment, and finally Section 7 draws conclusions.

2. Indoor Positioning Platforms

Several indoor positioning platforms have been proposed in recent years that have the potential to take into account IoT requirements and could be thus adopted within the IoT context. They include both large scale, enterprise range platforms supported by big Information Technology (IT) players and vendors [21,22], and smaller ones proposed by startups and/or emerging from academic research [23–25]. Among enterprise grade platforms, MazeMap [21] focuses on indoor maps, but offers a wide range of services, including indoor positioning, although details on how the positioning platform is designed are not available on the company website. Software Development Kits (SDKs) for Android and iOS are available on the MazeMap website. MeridianApps [22], owned by Aruba, provides a software platform for indoor navigation and positioning, integrating with Aruba Beacons and Tags [26]. In this case, as well, SDKs for mobile devices based on iOS and Android are available. Cisco integrates location-based services into its Connected Mobile Experiences (CMX) platform [27], now part of Cisco DNA Spaces, using both Bluetooth Low Energy and WiFi hardware, with 1-to-3 m accuracy for connected WiFi devices.

All the above platforms are characterized by a closed-source approach, with positioning algorithms running on company software, and not modifiable by users. This approach is also adopted by a few startups, such as GoIndoor [28] and InfSoft [24], both proposing a server-based positioning service based on closed-source software, and providing SDKs for iOS and Android.

Platforms designed and released by startups and academia are however often characterized by an open-source approach: a notable example is AnyPlace [23] that makes source code available on GitHub under the MIT open-source license. AnyPlace currently provides the code for the positioning server and apps for Android and Windows Phone devices. Indoor positioning in AnyPlace uses a combination of WiFi and inertial sensors on mobile devices, and relies on the cooperation between server and apps running on the devices.

The FIND platform [25] also makes source code available under a GNU General Public License. The platform is extremely flexible, as it provides both active and passive positioning, and can thus in theory locate any WiFi active device. In the case of active positioning, support is provided for Android devices as well as for several hardware platforms typically adopted in IoT, such as Raspberry Pi, Electric Imp, and Particle Photon. Sample code for each platform is provided on the FIND website. The FIND platform emerges as the most appealing solution for academic learning and research activities on indoor positioning in IoT, thanks to the combination of open-source software, support for several IoT hardware platforms, and availability of both active and passive positioning. The extension or improvement of positioning algorithms requires however significant work on both the positioning server code and possibly on the software interfaces for the supported platforms, often written in different programming languages, making for a steep learning curve.

As a summary, it can be concluded that existing indoor positioning platforms, when analyzed from the point of view of study and research on IoT applications, present one or more of the following limitations:

- proprietary software - most of enterprise grade indoor positioning platforms are characterized by proprietary software for both clients and server, preventing end users/customers from customizing and improving positioning algorithms and metrics [21,22];
- limited device support - although all platforms support smartphones, albeit in some cases only Android ones, only a few support other devices. A notable exception is the FIND platform [25], which supports several devices by dedicated interfaces and provides passive positioning of all active WiFi devices within the coverage area;
- lack of integration with IoT platforms - each indoor positioning platform provides its own apps, user interfaces and Application Programming Interfaces (APIs), making the integration in a IoT system rather challenging. Even in flexible platforms, such as FIND, the extension or improvement of positioning algorithms requires significant work on both the positioning server code and on the software interfaces for the supported platforms, often written in different programming languages.

3. Internet of Things Platforms and ThingSpeak

The growing interest in IoT has led to the release of hundreds of different platforms [29]. Proprietary platforms are typically characterized by more mature development environments and deployment options, which typically streamline and simplify the deployment of large scale IoT apps. These benefits come however most often at the price of significant license/subscription fees. Notable exceptions exist: both Amazon Web Services (AWS) [30] and Google Cloud IoT [31] provide a basic free access tier. From the point of view of learning and research in academia, however, platforms providing access to the source code under an open-source license should be preferred, since they provide the flexibility required for the development and testing of new indoor positioning algorithms. For this reason, the review below will focus specifically on open-source platforms.

- FIWARE [32] is an effort funded by the European Union to develop an open solution to IoT. The platform is centered around the FIWARE Context Broker, which provides an implementation of the Next Generation Sensors Initiative v2 (NGSIv2) REpresentational State Transfer (REST) API to support queries from sensors, issuing commands to actuators, and more in general to enable the management of IoT context information between hardware devices and applications. Several additional components are available under the form of "Generic Enablers", providing additional functions, e.g., connectivity with hardware. Protocols implemented with dedicated enablers include the Machine-To-Machine (M2M) specific Message Queuing Telemetry Transport (MQTT) protocol, Open Platform Communications-Unified Architecture (OPC-UA), LightWeightM2M (LWM2M), while integration is supported through JavaScript Object Notation (JSON) and UltraLight2.0 data formats. In terms of data processing and data visualization, the platform currently provides a few enablers for video stream editing.

- The Kaa platform [33] provides a standalone server installation for local execution, and supports remote execution for a fee on the AWS platform. Kaa supports currently about 15 hardware platforms by means of endpoint SDKs written in C, C++, Objective-C, or Java, depending on the hardware. The platform also provides an advanced User Interface (UI) for the management of clients and applications, and supports popular client/server communication protocols, including Hyper Text Transfer Protocol (HTTP), Extensible Messaging and Presence Protocol (XMPP) and MQTT. Furthermore, Kaa supports the transfer of data collected at the central server to a wide range of back-ends and databases, including Oracle, Apache, and MongoDB, where processing of data can be later carried out. Kaa does not offer however data processing capabilities on the platform itself, beyond the transformation of raw data into time series. The platform also provides a set of widgets that can be used to visualize data and to send commands to connected devices.
- Lelylan [34] is a platform composed by a set of micro-services that provide key functionalities, including connectivity between clients and server. Lelylan is particularly suited to connect a user with remote-controlled devices (e.g., light switches or actuators) using HTTP and MQTT protocols, and supports a wide range of hardware platforms (about 20) by means of so-called physical APIs. The platform does not provide however built-in data processing capabilities. Lelylan is in fact based on an event bus that registers user requests sent from a Web app over HTTP and makes them available to a Physical Proxy. The Physical Proxy forwards then the requests to the interested hardware devices for the execution of the actions correlated with the request itself. The platform also provides real time feedback and notifications to inform the user and/or other entities of the outcome of the actions.
- OpenMTC [35] is a reference implementation of the oneM2M standard for M2M communications, and provides APIs to connect hardware through HTTP, HTTPS, MQTT, and feed collected data to a middleware aggregator in charge of sending such data to external applications. Although interesting, the OpenMTC does not provide any built-in processing and visualization capabilities, relying on external apps to carry out these tasks.
- SiteWhere [36] is a platform available in both Community (free) and Enterprise (paid) editions, and similarly to Lelylan is organized in micro-services that provide connectivity with devices, integration with external services, and command delivery to devices. SiteWhere supports MQTT, Advanced Message Queuing Protocol (AMQP) and Streaming Text Oriented Messaging Protocol (STOMP) protocols for communications with devices, using JSON or Protocol Buffers data formats. No specific provisions are given regarding data processing and visualization, mainly relying on the integration with external databases for offline data processing in Azure, Apache SoIr, and other services.
- The ThingBox [37] is a Raspberry Pi-based IoT platform that takes advantage of the Node-Red visual editor [38] by IBM to develop IoT applications. Device support and processing capabilities are based on "nodes" made available in the Node-Red editor, each node corresponding to a specific feature (e.g., support for a specific hardware or software platform). The UI for management is however less developed than those provided by Kaa and Lelylan. Furthermore, The ThingBox is defined to be a "local" IoT implementation, with no cloud support. Although multiple Raspberry Pis can operate simultaneously, they will not interact.
- ThingSpeak [39] is an open-source IoT platform composed by a central server that provides data collection, processing and analysis, and libraries/APIs to support IoT devices. The open-source central server can be installed locally or run in the cloud, with both free and paid pricing schemes. ThingSpeak stands out for its extremely easy interface for data processing and presentation, thanks to the support of the Matlab language in the cloud deployment, including graphic output and several Matlab toolboxes. Access to such toolboxes is granted based on a paid MathWorks license in addition to basic, free Matlab support provided by ThingSpeak. The platform provides web APIs using both the HTTP and MQTT communication protocols, allowing the support of a wide range of devices, including smartphones as well as all major IoT hardware platforms, such as Arduino, Raspberry Pi, Electric Imp, Particle Photon, and ESP8266. Integration with external

services is possible by exporting data in JSON, Comma Separated Values (CSV) and eXtensible Markup Language (XML) formats, or by adopting the ThingHTTP protocol.

Table 1 provides a comparison of the open-source platforms discussed above according to the following criteria: support for communication protocols, data processing, data visualization, and integration with external services. Table 1 highlights that while most platforms provide satisfactory support for IoT communications protocols and integration with external services, ThingSpeak stands out for its unique features in terms of data processing and visualization.

Table 1. Qualitative comparison of open-source platforms analyzed in Section 3.

Platform	Communication Protocols	Data Processing	Data Visualization	Integration with External Services
FIWARE	Excellent	Sufficient	Sufficient	Excellent
Kaa	Excellent	Limited	Good	Excellent
Lelylan	Good	Limited	Limited	Good
OpenMTC	Good	Limited	Limited	Good
SiteWhere	Good	Limited	Limited	Excellent
The ThingBox	Sufficient	Sufficient	Sufficient	Good
ThingSpeak	Good	Excellent	Excellent	Good

The possibility of easily defining analysis and visualization apps within the platform provides in fact a significant advantage compared to all other open-source platforms. In addition, the built-in support for execution of Matlab scripts and functions within such apps in the cloud is extremely appealing, given the widespread use of Matlab in the academic environment, both in research and in undergraduate and graduate courses. The support for Matlab guarantees in fact an almost seamless transfer of data processing and analysis algorithms from simulation environments to experimental testbeds, thanks to the time and efforts spared for porting such algorithms from Matlab to other languages. Please note that Matlab support is also available in other platforms: Matlab can in fact run in a virtual machine in AWS through its S3 cloud service as well as in Microsoft Azure [40] and a few other platforms. This solution requires however a Matlab license with cloud support and in most cases a paid license for the selected cloud platform (this is the case for AWS, where a paid account is needed to access the S3 service). ThingSpeak, on the other hand, provides the possibility of executing Matlab code out of the box, with no additional license required. This feature made ThingSpeak the best candidate for the design of the ThingsLocate IoT indoor positioning platform for learning and research in academia. More details on ThingSpeak are thus provided in the following.

ThingSpeak relies on two key concepts for data exchange and storage: channels and messages.

- A channel is a data structure identified by a unique id, and by keys for writing to and reading from it, and is the basic structure for data storage in ThingSpeak, under the form of timestamped data entries. As an example, a channel could be used to store temperature readings by a sensor in time. Each entry in the channel would then include a reading with the corresponding timestamp. A channel can contain up to 8 fields of data, and an entry in a channel is defined as the combination of the 8 fields of data and of a timestamp.
- A message is the base information transfer unit, and is defined as the information transferred when an entry is written to a ThingSpeak channel by a device using the ThingSpeak write API.

ThingSpeak provides an extremely friendly interface for processing data written in channels by devices, under the form of ThingSpeak apps. Apps can be executed once, periodically, or as a reaction to a new message being written on a channel, and allow processing of the data, visualize them by generating graphs, or publish them on new channels or on the Internet by posting them on web pages and social networks. As already mentioned, data analysis, in particular, is made easy in ThingSpeak thanks to the possibility of creating Matlab analysis apps, consisting of Matlab scripts

that are stored on the platform and are executed on data uploaded on one or more ThingSpeak channels.

4. The ThingsLocate Platform

ThingsLocate provides an open-source solution for indoor positioning of WiFi IoT devices based on RSSI fingerprinting; information on the availability of the ThingsLocate platform is provided in Appendix A. Based on the analysis carried out in Section 3, ThingsLocate was developed on top of the ThingSpeak (TS) platform [39]. ThingsLocate takes advantage of ThingSpeak support for remote execution of Matlab code, allowing for easy addition and testing of new algorithms, most often originally developed in Matlab, without the overhead of porting them to other languages. This dramatically eases the learning curve for the use and extension of the platform, making it suitable for both research and graduate course teaching in an academic environment. From a deployment point of view, ThingsLocate is straightforward to adopt on the many existing devices supported by ThingSpeak, and it keeps most of the complexity in the cloud, leading to two key advantages: a) it is suitable for a very wide range of IoT devices with very limited processing capability, and b) it does not require any local processing infrastructure to be installed and maintained.

ThingsLocate uses RSSI data measured by a target IoT device and loaded to the platform using ThingSpeak APIs to determine the position of the device itself. The position information is then made available for visualization and processing to the device itself or to any interested entity. Figure 1 shows how the introduction of the ThingSpeak platform modifies the system architecture and the workflow for an indoor positioning request.

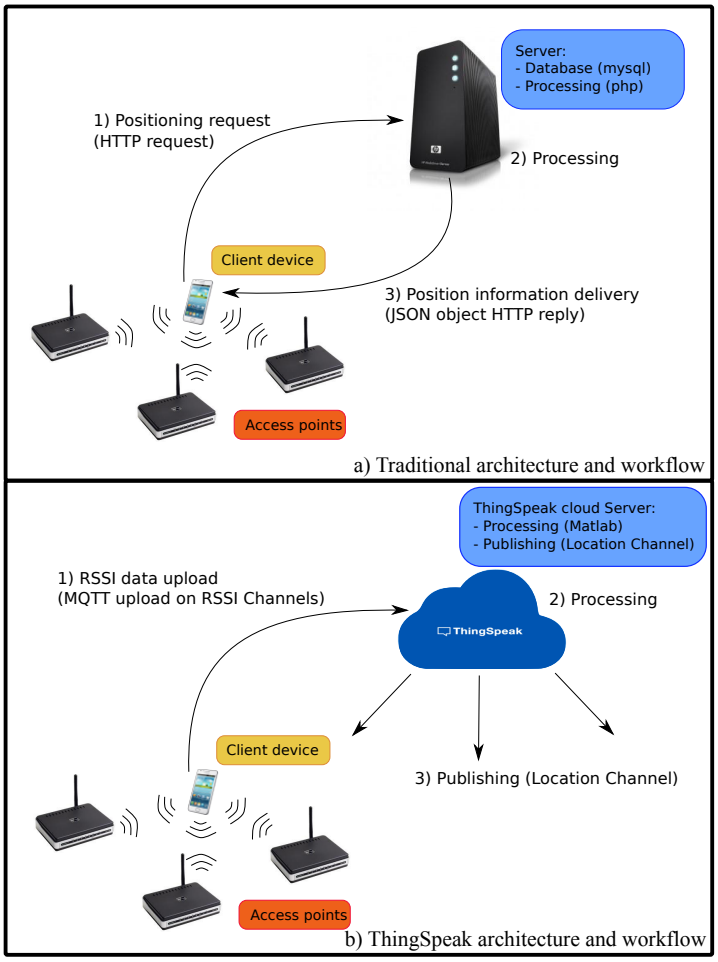

Figure 1. Comparison between traditional architecture and workflow (Figure **1a**) and cloud architecture and workflow in ThingsLocate (Figure **1b**).

To operate, ThingsLocate inherits the following TS elements:

- RSSI channels—A set of N_{RC} TS channels used by target devices to upload RSSI data. N_{RC} is a system parameter to be selected at platform deployment time, as explained later;
- Locate App—A TS Matlab Analysis app implementing the positioning algorithm, with execution triggered by the upload of data on the RSSI channels by a device;
- Location channel—A TS channel used to publish the estimated location.

A detailed description of the RSSI and Location channels, of the Locate App, as well as of the procedures to setup and use the ThingsLocate platform and of the client code to be used on ThingSpeak compatible devices is provided in the following subsections.

4.1. RSSI Channels

Table 2 presents the structure of the N_{RC} RSSI channels, labeled $RSSI\,1, \ldots, RSSI\,N_{RC}$, in the case $N_{RC} = 3$. $RSSI\,1$ is used to provide general information and settings on the positioning request, while the remaining channels are used to provide RSSI data. The data uploaded on channel $RSSI\,1$ include the ID of the IoT device *Device ID*, the number of detected APs N^D_{RSSI}, the positioning algorithm that the device requests and the corresponding settings, as detailed in Section 4.3. While the *Device ID* is a mandatory information, the remaining data pieces are optional and may be ignored if the IoT device does not possess the technical capability required to provide them. The *Device ID* information is repeated on each channel, to ensure that RSSI data coming from different target devices are not mistakenly mixed-up during position estimation.

Table 2. Fields of the RSSI channels, used by the IoT device during the Upload step, for $N_{RC} = 3$.

Channel	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8
$RSSI\,1$	*Dev. ID*	N^D_{RSSI}	*Algorithm*	*Param*1	*Param*2	*Param*3	*Param*4	*Param*5
$RSSI\,2$	*Dev. ID*	N/A	AP_1 *ID*	AP_1 *RSSI*	AP_2 *ID*	AP_2 *RSSI*	AP_3 *ID*	AP_3 *RSSI*
$RSSI\,3$	*Dev. ID*	N/A	AP_4 *ID*	AP_4 *RSSI*	AP_5 *ID*	AP_5 *RSSI*	AP_6 *ID*	AP_6 *RSSI*

4.2. Location Channel

The Location channel is used to publish the estimated position of devices that upload their measured RSSI data through the RSSI channels. Each entry in the channel corresponds to a position estimate. Six out of the eight available fields in the channel are used to provide the following information: (a) *Device ID* of the device that uploaded the RSSI data used for the estimation; (b) position of the device in a 3D system of coordinates (3 fields); (c) estimated floor in a multifloor building; (d) algorithm used for positioning. The remaining two fields are currently reserved for future development.

4.3. Locate App

The Locate App includes three components: (a) RSSI data collected during the offline phase, specific for each deployment, (b) a set of positioning algorithms and (c) a set of similarity metrics. The app adopts the selected combination of positioning algorithm and similarity metric to process the offline data and the information written by the target device in the RSSI channels to estimate the position of the device.

The offline RSSI data are stored in the App under the form of three Matlab variables:

- a bidimensional matrix $\boldsymbol{RSSI\_MAP}$ of size $N_{RP} \times N_{AP}$, where N_{RP} is the total number of RPs defined in the area covered by the positioning system, and N_{AP} is the total number of different APs detected in the area. Please note that in areas of large dimensions only a subset of the APs will be detected in each RP, and N_{AP} will be thus the size of the union of all APs detected in at least a single RP. For a generic AP i not detected at a generic RP j, the entry $\boldsymbol{RSSI\_MAP}_{i,j}$ will be set to a reference value P_{NULL} lower than the receiver sensitivity threshold;

- a bidimensional matrix ***RP_MAP*** of size $N_{RP} \times 3$, containing in j-th row the 3D coordinates of the j-th RP;
- a vector ***AP_ID*** of length N_{AP}, storing the MAC address of each detected AP in the same order adopted to fill in each row of the ***RSSI_MAP*** matrix;

Each positioning request is served by the Locate App by executing the following steps:

1. The pairs $\left\{ < AP_x\ ID, AP_x\ RSSI >, \quad x = 1, \ \ldots, \ N^D_{RSSI} \right\}$ loaded by the target device on the RSSI channels are mapped on a vector $\boldsymbol{s}_T$ of length N_{AP}, using the information contained in the ***AP_ID*** vector as follows: the generic j-th element of $\boldsymbol{s}_T$, $\boldsymbol{s}_{j,T}$, is set either to the RSSI value measured for $\boldsymbol{AP\_ID}_j$, if this was uploaded by the target device, or to P_{NULL}, otherwise;
2. The position of the target device is estimated from ***RSSI_MAP*** and $\boldsymbol{s}_T$ using either the combination of positioning algorithm and positioning metric requested by the device in the $RSSI$ 1 or the default one, if no requests are expressed;
3. the estimated position is published on the Location channel, following the structure described in Section 4.2.

A description of the algorithms and the metrics available on the platform is given in the following, where for brevity the vector corresponding to row i of the ***RSSI_MAP*** matrix is indicated with $\boldsymbol{s}_i$.

The set of algorithms currently implemented on the platform includes the k Nearest Neighbor (kNN) algorithm [41] and three enhanced versions of kNN. A key issue in kNN algorithms is the selection of k. With respect to this issue, algorithms can be broadly divided into two families: a priori fixed k algorithms, in which k does not depend on the positioning request [18,42,43], vs. dynamic k algorithms that determine the best k depending on the specific positioning request [44–46]. The set of algorithms implemented in ThingsLocate includes two examples for each family: *kNN* and the Weighted kNN (WkNN) algorithm [47] for fixed k, vs. the Enhanced Weighted K Nearest Neighbor (EWkNN) [44] and the frequentist inference K Nearest Neighbor (FI-WkNN), proposed in [46], for dynamic k. The four algorithms are introduced in the following.

- In *kNN*, the k RPs with RSSI values most similar to the ones recorded by the target device are selected, according to a similarity metric $m_i = m\left(\boldsymbol{s}_T, \boldsymbol{s}_i\right)$. The estimated position $\tilde{p} = \{\tilde{x}, \tilde{y}, \tilde{z}\}$ is then obtained as the average of the positions $p_1 = \{x_1, y_1, z_1\}, \ \ldots, \ p_k = \{x_k, y_k, z_k\}$ of the selected RPs:

$$\begin{aligned} \tilde{x} &= \frac{\sum_{n=1}^{k} x_n}{k} \\ \tilde{y} &= \frac{\sum_{n=1}^{k} y_n}{k} \iff \tilde{p}_{kNN} = \frac{\sum_{n=1}^{k} p_n}{k}. \\ \tilde{z} &= \frac{\sum_{n=1}^{k} z_n}{k} \end{aligned} \tag{1}$$

- *WkNN* is an extension of *kNN* in which the positions of the k selected RPs are weighted according to a weighting function $\mathrm{w}(\cdot)$ evaluated on the similarity metric m_n:

$$\tilde{p}_{WkNN} = \frac{\sum_{n=1}^{k} \mathrm{w}(m_n) p_n}{\sum_{n=1}^{k} \mathrm{w}(m_n)}. \tag{2}$$

  $\mathrm{w}(\cdot)$ is typically defined so to emphasize the weight of RPs that show higher similarity to the Test Point (TP); kNN can be considered to be a special case of WkNN where $\mathrm{w}(\cdot) = 1$ for any argument.
- *EWkNN*, introduced in [44], as mentioned before, determines k dynamically for each positioning request, as a result of a two-step pruning procedure of the set of RPs. Two strategies can be adopted in the selection of the thresholds to be used in the pruning steps: a static strategy in which the thresholds are predetermined, and a dynamic one, in which they depend on the content of $\boldsymbol{s}_T$.

- *FI-WkNN*, proposed in [46], relies on frequentist theory of inference combined with a measure of similarity given by the Pearson's correlation R statistical index. The algorithm uses the p-value probabilities associated with a test statistic T defined over R, as defined in frequentist inference, to determine the relevance of each RP: RPs characterized by a low p-value are deemed more relevant than those with a high p-value. FI-WkNN can work with either a fixed or a dynamic k selection strategy. In the former case the k RPs with the smaller p-values are considered in the estimation of the position of the target device using Equation (2), while in the latter case a RP j is included if it passes an hypothesis test defined as $p_j < \alpha$, where α is a tunable threshold.

The selected positioning algorithm can be used in combination with any of the similarity metrics available in the Locate App. The similarity metric in a fingerprinting positioning system is used to determine the set of k fingerprints that are most similar to the fingerprint contained in the $\boldsymbol{s}_T$ vector provided by the target device. This is typically achieved by determining the similarity between $\boldsymbol{s}_T$ and $\boldsymbol{s}_i$ (the i-th row of $\boldsymbol{RSSI\_MAP}$) for $i = 1, \ldots, N_{RP}$, sorting the N_{RP} rows according to decreasing similarity, and selecting the first k. The similarity metric plays thus a key role in determining the performance of a fingerprinting positioning system. ThingsLocate currently provides two options for the similarity metric:

- the inverse Minkowski distance, with order p, defined as follows:

$$m(\boldsymbol{s}_T, \boldsymbol{s}_i) = \left(\sum_{l=1}^{N_{AP}} |s_{l,T} - s_{l,i}|^p \right)^{-\frac{1}{p}} \quad p \geq 1. \tag{3}$$

  Equation (3) actually defines a family of metrics as a function of the order $p \geq 1$. The most common choices in the literature are $p = 1$, corresponding to the inverse of the Manhattan distance, and $p = 2$, corresponding to the inverse of the Euclidean distance.
- the squared value of the Pearson correlation coefficient $R(\boldsymbol{s}_T, \boldsymbol{s}_i)$, defined as:

$$R(\boldsymbol{s}_T, \boldsymbol{s}_i) = \frac{\sum_{l=1}^{N_{AP}} (s_{l,T} - \bar{\boldsymbol{s}}_T)(s_{l,i} - \bar{\boldsymbol{s}}_i)}{\sqrt{\sum_{l=1}^{N_{AP}} (s_{l,T} - \bar{\boldsymbol{s}}_T)^2} \sqrt{\sum_{l=1}^{N_{AP}} (s_{l,i} - \bar{\boldsymbol{s}}_i)^2}}. \tag{4}$$

  $m(\boldsymbol{s}_T, \boldsymbol{s}_i) = R^2(\boldsymbol{s}_T, \boldsymbol{s}_i)$, typically known as the coefficient of determination, is automatically used when the *FI-WkNN* [46] is selected since in this algorithm the similarity between fingerprints is determined on the basis of a statistical test on a test statistic based on $R^2(\boldsymbol{s}_T, \boldsymbol{s}_i)$. The metric can however be used in combination with any of the other algorithms, as also proposed in [46].

Additional similarity metrics can be easily introduced by adding the corresponding code in the Locate App. An overview of possible similarity metrics suitable for addition to the platform can be found in [19].

The specific settings to be used for the selected combination of algorithm and metric are provided in the *RSSI* 1 channel. Table 3 shows the parameter to be provided in each field of the channel. Please note that the expected data in some fields (and whether such fields are used or not) depend on the content of previous fields. Allowed values for each parameter listed in Table 3 are provided in Table 4.

As a final note, the WkNN-based algorithms in ThingsLocate use the selected similarity metric also as the weighting function, adopting thus $\mathrm{w}(x) = x \; \forall \; x$, following the most common approach in the literature; the combination of different similarity metrics vs. weighting functions was in fact only recently investigated [19].

Table 3. Content of fields 4 to 8 of the *RSSI*1 channel for the algorithms and metrics available in ThingsLocate.

Algorithm	Param1	Param2	Param3	Param4	Param5
*k*NN	*k*	Metric	*p* order/N/A	N/A	N/A
W*k*NN	*k*	Metric	*p* order/N/A	N/A	N/A
EW*k*NN	Metric	*p* order/N/A	Threshold strategy	Threshold 1/N/A	Threshold 2/N/A
FI-W*k*NN	*k* strategy	*k* / α	N/A	N/A	N/A

Table 4. Allowed values for parameters listed in Table 3.

Parameter	Details	Allowed values
k	Number of neighbors used in estimation	Any positive integer number
Metric	Similarity metric	{'*Minkowski*', '*Pearson*'}
p order	*p* order used in the Minkowski distance	Any positive integer number
Threshold strategy	Strategy for threshold definition in the EW*k*NN algorithm	{'*StaticThreshold*', '*DynamicThreshold*'}
Threshold 1	Value of first threshold in the EW*k*NN algorithm when 'StaticThreshold' strategy is adopted	Any positive real number
Threshold 2	Value of second threshold in the EW*k*NN algorithm when 'StaticThreshold' strategy is adopted	Any positive real number
k strategy	Strategy for selection of *k* in the FI-W*k*NN algorithm	{'*StaticK*', '*DynamicK*'}
α	Hypothesis test threshold used in the FI-W*k*NN algorithm when 'DynamicK' strategy is adopted	Any positive real number

4.4. ThingsLocate Setup Procedure

Assuming that the WiFi hardware is already installed in the area of interest, the procedure to setup and deploy the ThingsLocate platform foresees three steps: (1) *Environment scan*, (2) *Server-side configuration*, and (3) *Device-side configuration*. A summary of the procedure is presented in Figure 2, while a detailed description is provided in the following.

1. *Environment scan*, corresponding to the offline phase of WiFi fingerprinting introduced in Section 1, focuses on the acquisition of the data required to create the ***RSSI_MAP***, ***RP_MAP*** and ***AP_ID*** data structures. This step starts with the definition of the set of Reference Points and the recording of the corresponding coordinates in the ***RP_MAP*** matrix, as well as of the corresponding N_{RP} value. Next, RSSI data are collected at each RP. Once all measurements have been carried out, the union of AP IDs detected in each of the RPs is used to create the ***AP_ID*** vector, and determine thus the dimensions of the ***RSSI_MAP*** matrix. Finally, RSSI data are recorded in the ***RSSI_MAP*** following for the rows the same order adopted for ***RP_MAP***, and within each row the same order adopted for the ***AP_ID*** vector. The time required for this step is directly related to the number N_{RP} of RPs to be analyzed that, in turn, depends on the size of the area to be served by the positioning system and on the selected spatial density of RPs. In the case of large areas and/or high densities of RPs, the *Environment scan* step can require hours or days of work, and is by far the most time-consuming step in the setup procedure. This issue is inherent to the WiFi fingerprinting approach and not to its implementation in ThingsLocate. A possible solution will be discussed in Section 6.
2. *Server-side configuration* focuses on the preparation of the cloud elements of the ThingsLocate platform. It includes the creation of the RSSI and Locate channels and the configuration of the Locate App, based on the information collected during the *Environment scan* step and on the channel IDs of the RSSI and Locate channels just created. The number N_{RC} of RSSI channels to be created is also determined in this step, based on the data collected in the *Environment scan* step.

Since each channel can contain the data for three APs, the number of RSSI channels to be created is approximately equal to one third of the maximum number of APs detected in a single RP.

3. *Device-side configuration* deals with the configuration of client code in each device. It consists of writing the IDs of the RSSI and Location channels and the value of N_{RC}, needed by the device to determine how many RSSI values can be reported when submitting a positioning request.

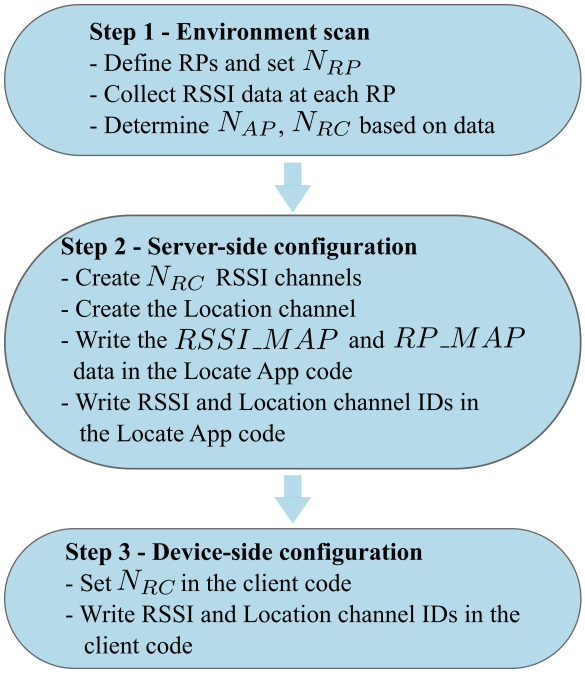

Figure 2. Setup and deployment procedure for the ThingsLocate platform.

4.5. Positioning Procedure

The procedure used to determine the position of an IoT target device is divided into two steps:

1. *Upload*—the target device collects RSSI data as pairs $\{ID, RSSI\}$, and uploads them on the *RSSI* channels, jointly with control information as previously described;
2. *Locate*—the information upload at Step 1 triggers the execution of the Locate Analysis app that reads data from the RSSI channels, estimates the position of the IoT device using the selected algorithm, and writes the estimate on the *Location* TS channel.

Additionally, the write operation at the end of Step 2 may trigger the execution of publishing actions available on TS, such as email delivery or tweeting on Twitter.

4.6. Client Code

The client code made available in the current release of the ThingsLocate platform covers Linux, MacOS, and Android devices, as well as Raspberry Pi devices. The extension to other devices supported by ThingSpeak is planned for the next future. The client code for all supported devices shares a common set of features, briefly described in the following.

A client device is required to perform two main tasks:

1. collection of RSSI data during the offline phase (*Environment scan* step);
2. collection of RSSI data and submission of positioning requests during the online phase.

The same code is used to perform the two tasks. The only difference is that during the collection of data for the *Environment scan*, RSSI data are stored locally on the device (since RSSI channels are typically not yet available), while during the online phase they are uploaded on the RSSI channels. To this aim, the code provides the following features:

- configurable number of measurements to be averaged, to increase reliability of the RSSI data;
- configurable filename for offline storing (for the offline phase);
- automatic ordering of collected data in decreasing order of RSSI value, so that if more than $3 * N_{RC}$ values are collected the less important ones are discarded;
- configurable positioning algorithm, positioning metric and corresponding settings to be uploaded on the $RSSI$ 1 channel (for the online phase).

An example of the configuration screen for the Android app is presented in Figure 3.

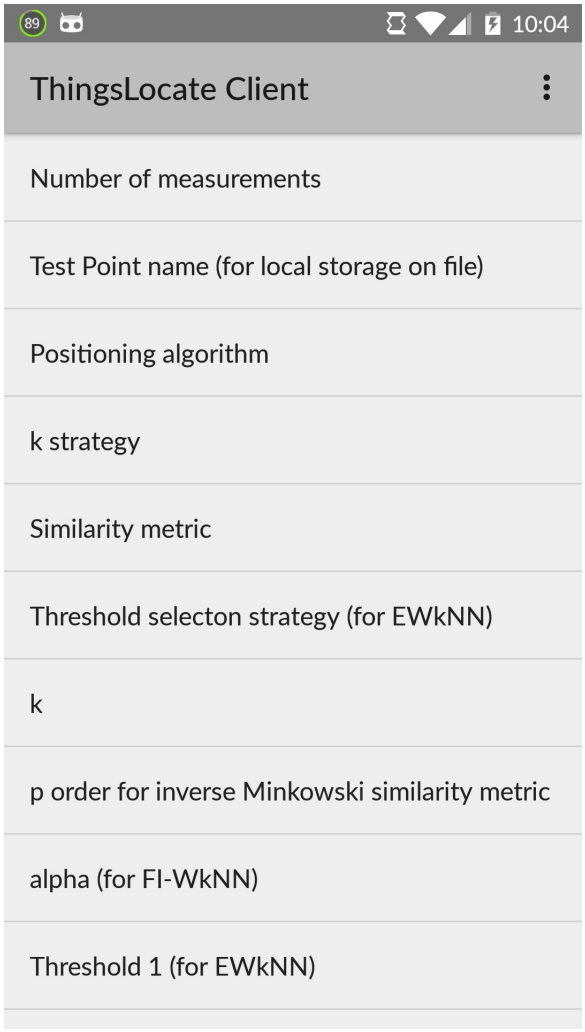

Figure 3. Configuration screen for ThingsLocate Android app.

5. Proof of Concept: Indoor Positioning of Android and Raspberry Pi Devices

In this Section a proof-of-concept implementation of the ThingsLocate platform is proposed, in order to illustrate how the setup procedure define in the previous Section works in a real-world environment, and to highlight the capabilities offered by the platform for learning and research on indoor positioning for IoT. Section 5.1 illustrates settings and results of the setup procedure, while Sections 5.2 and 5.3 provide an example of the experimental campaigns that can be carried out using ThingsLocate. Two issues were investigated using the TL platform: (a) the impact of different devices on positioning accuracy, presented in Section 5.2, and (b) the impact of positioning algorithms, presented in Section 5.3.

Please note that the goal of this Section is not to provide an exhaustive analysis of the two issues, but rather to prove that a similar analysis, typically of great interest in academic research and graduate studies, is feasible and easy to perform using ThingsLocate with limited setup efforts.

5.1. Setup

The ThingsLocate platform was deployed at the second floor of the Department of Information Engineering, Electronics, and Telecommunications (DIET) of Sapienza University of Rome, Rome, Italy, covering a total area of approximately 42 × 12 m^2. The deployment took advantage of the testbed previously deployed in this location, described in detail in [48]. $N_{RP} = 72$ RPs were identified, with an average distance between two RPs of approximately 3 m. In each RP $q = 50$ samples were averaged to counteract the impact of the channel variability on the fingerprinting database. Each sample consists of the RSSI values received from all detected APs. Data were collected using an Apple Macbook Pro 15 Early 2011 (Apple Inc., Cupertino, CA, USA), equipped with a Broadcom BCM94322 2.4 Ghz/5 GHz WiFi card (Broadcom Inc., San Jose, CA, USA). The offline fingerprint database includes data collected for 6 APs at both 2.4 GHz and 5 GHz and 7 at 2.4 GHz, for a total of 19 WiFi signals. As a result, of the deployment procedure described in Section 4.4, the following settings were adopted: $N_{RP} = 72$, $N_{AP} = 19$, $N_{RC} = 13$ corresponding to a maximum of $3 \times (N_{RC} - 1) = N_{AP}^{MAX} = 36$ RSSI measurements per request. Figure 4 shows a map of the area and the positions of both APs and RPs.

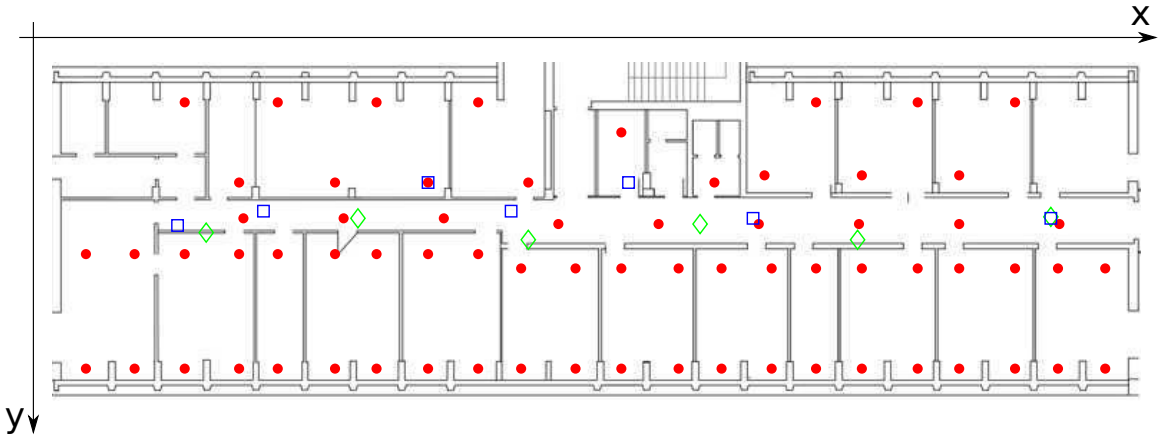

Figure 4. Positions of the RPs (filled red circles) and APs located at lower floor (green diamonds) and at same floor (blue squares) of the RPs in the ThingsLocate testbed.

A set of $N_{TP} = 10$ Test Points (TPs) was identified, and RSSI data were collected at each TP with three different devices: (1) a Raspberry Pi B+ (Raspberry Pi Foundation, Cambridge, England) using a USB WiFi adapter based on a RealTek RTL8188CUS chipset (Realtek Semiconductor Corp., Hsinchu, Taiwan) operating at 2.4 Ghz, (2) a OnePlus One smartphone (OnePlus Technology Co., Shenzhen, China) with Android 6, equipped with a Qualcomm WCN3680 2.4 Ghz/5 GHz WiFi chip (Qualcomm Incorporated, San Diego, CA, USA), and (3) the same Macbook Pro used to collect data during the *Environment scan* step. Measurements were collected using the Python scripts and the Android app made available as part of the ThingsLocate software package for each of the three devices. As already mentioned in Section 4.6, the ThingsLocate client code for all platforms is designed so that in case more than N_{AP}^{MAX} APs are detected, the N_{AP}^{MAX} with highest RSSI are uploaded, in order to minimize the impact of dropping the exceeding measurements.

5.2. Impact of Devices on Positioning Accuracy

The analysis on the impact of devices on positioning accuracy was carried out with TL settings presented in Table 5.

Table 5. Settings adopted in fields 4 to 8 of the *RSSI* 1 channel for the analysis of the impact of different devices on positioning accuracy.

Algorithm	Param1	Param2	Param3	Param4	Param5
*k*NN	3	'*Minkowski*'	2	N/A	N/A
W*k*NN	3	'*Minkowski*'	2	N/A	N/A
EW*k*NN	'*Minkowski*'	2	'*DynamicThreshold*'	N/A	N/A
FI-W*k*NN	'*StaticK*'	3	N/A	N/A	N/A

Figure 5 presents the average positioning error observed for the N_{TP} TPs using the three devices introduced in Section 4.4, while Figure 6 shows the positioning error along the X vs. Y axes. In general, a lower positioning accuracy along the Y axis was observed for all devices, with errors on Y axis up to 2.2 times the errors observed on the X axis. This result can be explained by the suboptimal placement of APs, mostly transmitting from the central hall, with a very low spatial diversity along the vertical Y axis, as visible in Figure 4. Results in Figure 5 also show that the positioning accuracy observed for both Raspberry Pi and Android devices was lower than for the Macbook Pro. Furthermore, the Raspberry Pi was characterized by the worst accuracy for all the considered algorithms. Two factors contributing to the performance gap between the Macbook Pro and the other devices can be identified. The first factor is the mismatch between the device used for offline data collection and online measurements. The Macbook Pro can be considered in this scenario as a best-case benchmark, since for this device there is no impact from device mismatch between RPs and TPs. The positioning accuracy for the other considered devices may be affected by the fact that different devices typically report different RSSI values at the same location, due to multiple reasons. A first reason is the different sensitivity of the WiFi chipsets, especially when the devices are placed at the border of the coverage area of an AP. A second reason is the way RSSI is evaluated by the firmware/device driver: RSSI is in fact a somewhat arbitrary representation of signal intensity, and different WiFi chipset makers can apply different algorithms to obtain a numerical representation of such intensity. Such algorithms often include nonlinear transformations, such as clipping to a minimum or maximum value, that may affect the similarity between the offline fingerprints and the online data. Finally, in the specific case of the WiFi chipset used for the Raspberry Pi, a third reason is the lack of support for 5 GHz networks; however, experiments repeated excluding 5 GHz signals for all devices led to extremely similar results, indicating that the lack of 5 GHz support did not play a significant role.

The second factor to be taken into account, in particular to explain the markedly worse performance of Raspberry Pi with respect to the Android device, is related to the statistical properties of the collected RSSI values. Figure 7 presents the estimated Probability Density Function (PDF) of the RSSI values for the three devices across all the TPs, and shows how the Raspberry Pi is characterized by RSSI values spanning over a narrower range than the Android and Macbook Pro devices. The PDF for the Raspberry Pi also shows a strong peak around -75 dBm that is absent in the RSSI PDF of the other devices. The different shape and domain of the RSSI PDF for the Raspberry Pi leads to a significantly lower variance compared to Android and Macbook Pro, despite a similar mean value, as shown in Table 6. Please note that the variance shown here is measured across TPs: a lower variance means thus that the device tends to report similar RSSI values across all TP positions, and in turn that data collected at different TPs will "look" similar when a similarity metric is applied to compare them against the fingerprints in the offline database. Interestingly, a comparison between Figure 5 and Table 6 suggests indeed that positioning accuracy in WiFi fingerprinting is inversely related to RSSI variance across TPs. Further studies are however required to confirm this finding, in particular by separating the impact of device mismatch vs. RSSI distribution.

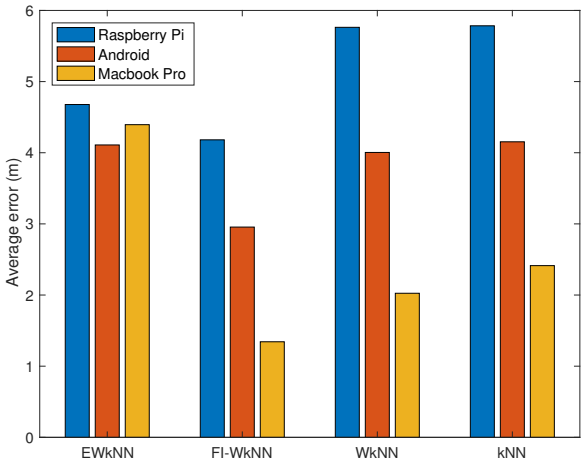

Figure 5. Average error observed for Raspberry Pi, Android, and Macbook Pro devices using the *kNN*, *WkNN*, *EWkNN* and *FI-WkNN* algorithms.

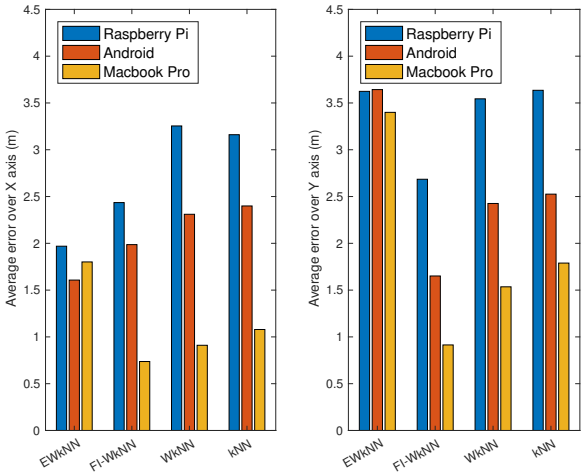

Figure 6. Average error along the *X* vs. *Y* axes observed for Raspberry Pi, Android, and Macbook Pro devices using the *kNN*, *WkNN*, *EWkNN* and *FI-WkNN* algorithms.

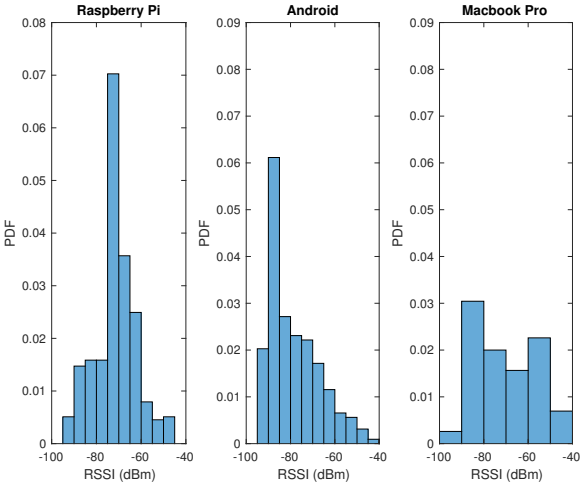

Figure 7. Probability density function of RSSI estimated from measurements collected in TPs for Raspberry Pi, Android, and Macbook Pro devices.

Table 6. Average value and variance of RSSI samples for the three devices used in the ThingsLocate testbed.

Device	RSSI Average Value (dBm)	RSSI Variance (dBm^2)
Raspberry Pi	−71.69	90.09
Android	−78.32	141.09
Macbook Pro	−69.94	204.13

5.3. Impact of Algorithms on Positioning Accuracy

The results in Figures 5 and 6 show that the different algorithms implemented in the TL platform may lead to significantly different results in terms of accuracy. The *EWkNN* algorithm, in particular, proved to be more sensitive to the suboptimal AP placement on the Y axis, leading to extremely high errors for all devices. On the opposite, the *FI-WkNN* consistently led to the best performance for all devices, significantly mitigating the negative impact of suboptimal placement of APs along the Y axis. One might wonder whether the performance of the algorithms is related to the specific settings adopted in the experiment presented in Section 5.2, and how algorithms would behave as a function of their parameters. The TL platform provides an ideal framework to compare the different algorithms under the same conditions. An example is presented here, showing an analysis of accuracy as a function of k for the Macbook Pro device. Results of this analysis, carried out with TL settings shown in Table 7, are presented in Figure 8, showing the average positioning error for the 4 algorithms available in the TL platform.

Table 7. Settings adopted in fields 4 to 8 of the *RSSI* 1 channel for the analysis of the impact of different algorithms on positioning accuracy.

Algorithm	Param1	Param2	Param3	Param4	Param5
*k*NN	1 − 10	"Minkowski"	2	N/A	N/A
W*k*NN	1 − 10	"Minkowski"	2	N/A	N/A
EW*k*NN	"Minkowski"	2	"DynamicThreshold"/"StaticThreshold"	N/A/100	N/A/20
FI-W*k*NN	"StaticK"	1 − 10	N/A	N/A	N/A

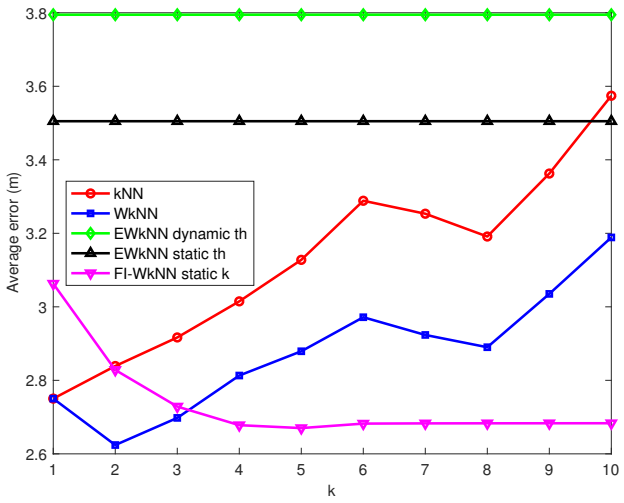

Figure 8. Average error for the *kNN*, *WkNN*, *EWkNN* and *FI-WkNN* using Macbook Pro TPs in the TL platform.

The *EWkNN* was tested using both dynamic thresholds and optimal static thresholds determined through heuristic search. Results show that although *WkNN* and *FI-WkNN* achieve similar best accuracy when the optimal value of k is adopted, *FI-WkNN* is more robust to suboptimal selection of k, in agreement with [46]; this is an important quality in a *WkNN*-based algorithm, given the impossibility of determining the optimal value of k in advance [19]. Results also confirm that the *EWkNN* leads to a poor overall performance, even with optimal thresholds, due to the sensitivity to errors on the Y axis, as shown for all devices in Figure 6.

6. Current Limitations and Future Developments

The adoption of the ThingSpeak cloud service as the basis for the development of the TL platform allows inheritance of the advantages provided by ThingSpeak in terms of device compatibility and native support for Matlab code, but also requires to deal with the specific limitations of this cloud service. In particular, the following limitations can be identified:

- limited update frequency and total messages—in its free pricing tier, ThingSpeak limits the frequency of updates on a channel to 1/15 s, and the total number of messages for an account to 3 million/year. The former limitation can in particular become a serious concern when many devices report their RSSI readings simultaneously, leading to an aggregate update frequency above the threshold. This issue can however be addressed in at least two ways: (1) subscribe to a different pricing tier that allows removal of this limitation by paying a monthly fee; (2) duplicate the set of RSSI channels by replicating their structure and distribute the devices over the different sets, in order to guarantee that the aggregate update frequency for each set is below the threshold.
- limited number of fields per channel—the channel structure in ThingSpeak foresees 8 fields, typically insufficient to upload all data required for a positioning request. The solution adopted in the TL platform, consisting of splitting the data over N RSSI channels, overcomes the issue, at the price however of a loss of efficiency in the use of the channels, since the repetition of the *Device ID* on each channel limits the number of AP readings that can be reported on each channel to 3. Although programming approaches have been proposed to force more than 8 fields in association with each channel, there is currently no straightforward solution to this issue.

Even taking into account the above limitations, the TL platform can be easily extended beyond its current capabilities, providing a starting point for the introduction of new features in the framework of a course assignment or a research project on indoor positioning for IoT. A partial list of possible developments includes:

- definition of new fingerprinting positioning algorithms and similarity metrics—new algorithms and metrics can be added in a straightforward manner, by adding them to the Locate app code without requiring any change to the client software. Any variation of *WkNN*, in particular, can be added with almost no coding effort.
- optimization of offline phase RSSI data collection—it was pointed out in Section 4.4 that the RSSI data collection carried out during the offline phase in the *Environment scan* step accounts for most of the time required for the setup procedure, and that this issue is inherent to the way WiFi fingerprinting works. Recently, however, a promising technique to solve this issue was proposed by the authors of the present work [48]. The technique, named *virtual fingerprinting*, allows reduction of the number of RPs to be processed (and the corresponding time) by at least one order of magnitude by generating synthetically the remaining ones using a deterministic channel model [49]. The integration of the technique would not require any change to the ThingsLocate platform, and would dramatically speed up the deployment of the platform.
- extension to WiFi positioning algorithms beyond fingerprinting—algorithms that rely on RSSI levels (e.g., trilateration algorithms based on power, as in [50]) can be added again by modifying accordingly the Locate app, without changing the structure of RSSI channels or the client code. The addition of algorithms that rely instead on different information, such as Time-Of-Arrival (TOA), Time-Difference-Of-Arrival (TDOA) or Direction-Of-Arrival (DOA), will typically require changes to both client code and Locate App. Note however that TOA, TDOA, and DOA require specific hardware support in client devices, making them typically less suitable for off-the-shelf IoT devices [14].
- adoption of other wireless technologies—the ThingSpeak platform can be easily adapted for the use of a wireless technology different from WiFi. The Locate App and the TS channels need no modifications, and the client code can be easily tweaked to collect RSSI data for the selected technology instead of WiFi. The use of another technology in place of WiFi is appealing mainly for

energy consumption reasons. WiFi is in fact a relatively energy-hungry technology, and frequent scans could quickly drain energy reserves in IoT devices not equipped with high capacity batteries. This issue could be addressed by adopting low-consumption RATs proposed for IoT devices, such as Bluetooth Low Energy, LoRa, and SigFox, at the price however of additional efforts to set up the wireless environment so that enough wireless signals are available for RSSI positioning. A rich wireless environment is in fact a key requirement for accurate RSSI-based positioning, and no other technology can currently match the spatial signal density of WiFi, typically guaranteeing tens or even hundreds of signals in an urban location.

7. Conclusions

This paper introduced ThingsLocate, a novel platform for WiFi-based indoor positioning of Internet of Things devices built upon the ThingSpeak IoT cloud platform. ThingsLocate fills the gap between IoT and indoor positioning and provides an easy-to-use and flexible tool for study and research of indoor positioning for IoT and location-aware IoT applications, by (a) avoiding the need for a local positioning server and of dedicated libraries for collecting and loading WiFi fingerprinting data, (b) allowing for easy development of new positioning algorithms thanks to the support of remote Matlab code execution offered by ThingSpeak, and (c) making available client-side scripts and apps for multiple devices. The smooth learning curve guaranteed by Matlab support and the user-friendly ThingSpeak user interface make ThingsLocate also very suitable for adoption in graduate courses on IoT-related topics.

The possibilities offered by ThingsLocate were shown in a proof-of-concept implementation, used to investigate the impact of device diversity and of different positioning algorithms on positioning accuracy. Preliminary results suggest that positioning accuracy is influenced by the variance of collected RSSI measurements provided by different devices, and that the impact of device diversity can be mitigated by the *FI-WkNN* algorithm, characterized by a metric different from the Euclidean distance typically adopted in *WkNN*-like algorithms. Future research work will focus on the extension of ThingSpeak so to support more hardware platforms and more advanced positioning algorithms, and on the integration of the virtual fingerprinting technique to streamline the setup procedure.

As a final note, although ThingsLocate was introduced in this work for research and learning purposes, it can also be adopted as a real-world solution for indoor positioning of IoT devices. The possibility of operating without any local processing infrastructure is in fact appealing for small shops and enterprises that do not have the skills and resources to host and maintain a local processing server: a ThingsLocate deployment could provide an indoor positioning service in a matter of hours without any hardware installation. Two aspects should however be taken into account when considering a real-world deployment of ThingsLocate. First, a reliable estimate of the number of devices to be located and of the corresponding generated traffic should be available, to determine the correct price tier to be acquired, as discussed in Section 6. Second, the location update period in a RSSI-based fingerprinting is inherently limited by the time required to perform one or more WiFi environment scans, typically in the order of minutes. Consequently, in its current form ThingsLocate is suitable for application scenarios involving portable or low-mobility devices, but not for scenarios requiring fast updates due to high mobility or abrupt position change detection (e.g., in anti-theft surveillance systems). A set of guidelines to evaluate whether ThingsLocate is a suitable solution for a given application scenario will be added to the platform documentation as part of future development activities.

Author Contributions: Investigation, L.D.N. and G.C.; Software, L.D.N.; Supervision, L.D.N. and M.G.D.B.; Writing—original draft, L.D.N.; Writing—review and editing, L.D.N., G.C. and M.G.D.B.

Funding: The work of Luca De Nardis and Maria Gabriella di Benedetto was supported by Sapienza University of Rome within research projects with Grants No. RP11715C7EFAA443, RP11715C7CA5279D, RP11816433F508D1, and RM116155068578FB. The work of Giuseppe Caso was partially supported by the H2020 5Genesis project, Grant Agreement No. 815178.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AMQP	Advanced Message Queuing Protocol
AP	Access Point
API	Application Programming Interface
AWS	Amazon Web Services
CIR	Channel Impulse Response
CMX	Connected Mobile Experiences
CSV	Comma Separated Value
DOA	Direction-Of-Arrival
GNSS	Global Navigation Satellite Systems
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol - Secure
IoT	Internet of Things
IT	Information Technology
JSON	JavaScript Object Notation
kNN	k-Nearest Neighbor
LWM2M	LightWeightM2M
M2M	Machine To Machine
MQTT	Message Queuing Telemetry Transport
NGSI	Next Generation Sensors Initiative
OPC-UA	Open Platform Communications-Unified Architecture
PDF	Probability Density Function
RAT	Radio Access Technology
REST	REpresentational State Transfer
R&D	Research and Development
RP	Reference Point
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
STOMP	Streaming Text Oriented Messaging Protocol
TDOA	Time-Difference-Of-Arrival
TL	ThingsLocate
TOA	Time-Of-Arrival
TP	Test Point
TS	ThingSpeak
UI	User Interface
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
WkNN	Weighted k-Nearest Neighbor

Appendix A

The ThingsLocate platform is freely available under a GNU AGPLv3 open-source license as a GitHub repository at the following address: https://github.com/lucadn/ThingsLocate. The repository includes the Locate App code, client scripts supporting MacOS, Linux/Raspberry Pi, and Android devices, and detailed instructions on how to setup and deploy the platform.

References

1. Nester, R. Internet of Things (IoT) Market : Global Demand, Growth Analysis and Opportunity Outlook 2023. Available online: https://goo.gl/XxRSd1 (accessed on 15 July 2019).
2. Markets and Markets. Internet of Things Technology Market by Hardware (Processor, Sensor, Connectivity Technology), Platform (Device Management Platform, Application Management Platform, Network Management Platform) Software Solutions, and Services, Application, and Geography-Forecast to 2022. Available online: http://www.marketsandmarkets.com/Market-Reports/iot-application-technology-market-258239167.html (accessed on 15 July 2019).
3. Juniper Research. THE INTERNET OF THINGS The Internet of Things—Consumer, Industrial and Public Services 2016–2021. Available online: https://www.juniperresearch.com/researchstore/key-vertical-markets/internet-of-things/consumer-industrial-public-services (accessed on 15 July 2019).
4. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context Aware Computing for The Internet of Things: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 414–454.
5. Al-Turjman, F.; Alturjman, S. Context-Sensitive Access in Industrial Internet of Things (IIoT) Healthcare Applications. *IEEE Trans. Ind. Inform.* **2018**, *14*, 2736–2744. doi:10.1109/TII.2018.2808190.
6. Guerrero-ibanez, J.A.; Zeadally, S.; Contreras-Castillo, J. Integration challenges of intelligent transportation systems with connected vehicle, cloud computing, and internet of things technologies. *IEEE Wirel. Commun.* **2015**, *22*, 122–128, doi:10.1109/MWC.2015.7368833.
7. Rong, W.; Vanan, G.T.; Phillips, M. The internet of things (IoT) and transformation of the smart factory. In Proceedings of 2016 International Electronics Symposium (IES), Denpasar, Indonesia, 29–30 September 2016; pp. 399–402, doi:10.1109/ELECSYM.2016.7861039.
8. Li, D.; Zhang, B.; Li, C. A Feature-Scaling-Based *k*-Nearest Neighbor Algorithm for Indoor Positioning Systems. *IEEE Internet Things J.* **2016**, *3*, 590–597.
9. Chen, C.; Chen, Y.; Han, Y.; Lai, H.Q.; Zhang, F.; Liu, K.J.R. Achieving Centimeter-Accuracy Indoor Localization on WiFi Platforms: A Multi-Antenna Approach. *IEEE Internet Things J.* **2017**, *4*, 122–134.
10. Chen, C.; Chen, Y.; Han, Y.; Lai, H.Q.; Zhang, F.; Liu, K.J.R. Achieving Centimeter-Accuracy Indoor Localization on WiFi Platforms: A Frequency Hopping Approach. *IEEE Internet Things J.* **2017**, *4*, 111–121.
11. Lee, C.H. Location-Aware Speakers for the Virtual Reality Environments. *IEEE Access* **2017**, *5*, 2636–2640.
12. Vasilateanu, A.; Goga, N.; Guta, L.; Mihailescu, M.N.; Pavaloiu, B. Testing Wi-Fi and bluetooth low energy technologies for a hybrid indoor positioning system. In Proceedings of IEEE International Symposium on Systems Engineering (ISSE), Edinburgh, UK, 3–5 October 2016; pp. 1–5.
13. Ciftler, B.S.; Kadri, A.; Guvenc, I. IoT Localization for Bistatic Passive UHF RFID Systems With 3-D Radiation Pattern. *IEEE Internet Things J.* **2017**, *4*, 905–916.
14. Figueiredo e Silva, P.; Kaseva, V.; Lohan, E.S. Wireless Positioning in IoT: A Look at Current and Future Trends. *Sensors* **2018**, *18*, doi:10.3390/s18082470.
15. De Angelis, G.; De Angelis, A.; Pasku, V.; Moschitta, A.; Carbone, P. A hybrid outdoor/indoor Positioning System for IoT applications. In Proceedings of IEEE International Symposium on Systems Engineering, Rome, Italy, 28–30 September 2015; pp. 1–6.
16. Wi-Fi Device Shipments to Surpass 15 Billion by End of 2016. Wi-Fi Alliance. Available online: https://www.wi-fi.org/news-events/newsroom/wi-fi-device-shipments-to-surpass-15-billion-by-end-of-2016(accessed on 15 July 2019).
17. Honkavirta, V.; Perala, T.; Ali-Loytty, S.; Piche, R. A comparative survey of WLAN location fingerprinting methods. In Proceedings of the 6th Workshop on Positioning, Navigation and Communication (WPNC), Hannover, Germany, 19 March 2009; pp. 234–251.
18. Bahl, P.; Padmanabhan, V.N. RADAR: an in-building RF-based user location and tracking system. In Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Tel Aviv, Israel, 26–30 March 2000; Volume 2, pp. 775–784.
19. Caso, G.; De Nardis, L.; Di Benedetto, M.G. A Mixed Approach to Similarity Metric Selection in Affinity Propagation-Based WiFi Fingerprinting Indoor Positioning. *Sensors* **2015**, *15*, 27692–27720.
20. Hossain, M.S. Cloud-Supported Cyber–Physical Localization Framework for Patients Monitoring. *IEEE Syst. J.* **2017**, *11*, 118–127.

21. MazeMap. MazeMap—Indoor Maps & WayFinding. Available online: https://www.mazemap.com (accessed on 15 July 2019).
22. MeridianApps BluDot. Available online: http://meridianapps.com (accessed on 15 July 2019).
23. Georgiou, K.; Constambeys, T.; Laoudias, C.; Petrou, L.; Chatzimilioudis, G.; Zeinalipour-Yazti, D. Anyplace: A Crowdsourced Indoor Information Service. In Proceedings of the 16th IEEE International Conference on Mobile Data Management (MDM '15), Pittsburgh, PA, USA, 15–18 June 2015; Volume 2, pp. 291–294.
24. InfSoft—Smart Onnected Locations. Available online: https://www.infsoft.com (accessed on 15 July 2019).
25. FIND. FIND—The Framework for Internal Navigation and Discovery. Available online: https://www.internalpositioning.com (accessed on 15 July 2019).
26. Avaliable online: https://www.arubanetworks.com (accessed on 15 July 2019).
27. Cisco Connected Mobile Experiences (CMX). Now Part of Cisco DNA Spaces. Available online: https://www.cisco.com/c/en/us/solutions/enterprise-networks/dna-spaces/index.html (accessed on 15 July 2019).
28. GoIndoor Navigation Technology. Available online: https://www.goindoor.co (accessed on 15 July 2019).
29. Singh, K.J.; Kapoor, D.S. Create Your Own Internet of Things: A survey of IoT platforms. *IEEE Consum. Electron. Mag.* **2017**, *6*, 57–68.
30. Amazon Web Services. Available online: https://aws.amazon.com (accessed on 15 July 2019).
31. Google IoT Core. Available online : https://cloud.google.com/solutions/iot-overview (accessed on 15 July 2019).
32. FIWARE. Available at https://www.fiware.org (accessed on 15 July 2019)
33. Kaa Project. Available online: http://www.kaaproject.org (accessed on 15 July 2019).
34. Lelylan. Available online: http://www.lelylan.com (accessed on 15 July 2019).
35. OpenMTC. Available onlie: http://www.open-mtc.org (accessed on 15 July 2019).
36. SiteWhere. Available online: http://www.sitewhere.io/ (accessed on 15 July 2019).
37. Thingbox. Available online: http://thethingbox.io/ (accessed on 15 July 2019).
38. Node-Red Visual Editor. Available online: https://nodered.org (accessed on 15 July 2019).
39. ThingSpeak. Available online: http://www.thingspeak.com (accessed on 15 July 2019).
40. Microsoft Azure IoT Accelerators. Available online: https://azure.microsoft.com/en-us/features/iot-accelerators/ (accessed on 15 July 2019).
41. Loftsgaarden, D.O.; Quesenberry, P.C. A nonparametric estimate of a multivariate density function. *Ann. Math. Stat.* **1965**, *36*, 1049–1051.
42. Li, B.; Salter, J.; Dempster, A.G.; Rizos, C. Indoor positioning techniques based on wireless LAN. In Proceedings of the 1st IEEE International Conference on Wireless Broadband and Ultra Wideband Communications, Waltham, MA, USA, 13–16 March 2006; pp. 13–16.
43. Yu, F.; Jiang, M.; Liang, J.; Qin, X.; Hu, M.; Peng, T.; Hu, X. 5G WiFi Signal-Based Indoor Localization System Using Cluster k-Nearest Neighbor Algorithm. *Int. J. Distrib. Sens. Netw.* **2014**, *10*, 1–12.
44. Shin, B.; Lee, J.H.; Lee, T.; Seok, K.H. Enhanced weighted K-nearest neighbor algorithm for indoor Wi-Fi positioning systems. In Proceedings of the 8th International Conference on Computing Technology and Information Management (NCM and ICNIT), Seoul, South Korea, 24–26 April 2012; pp. 574–577.
45. Philipp, M.; Kessel, M.; Werner, M. Dynamic nearest neighbors and online error estimation for SMARTPOS. *Int. J. Adv. Internet Technol.* **2013**, *6*, 1–11.
46. Caso, G.; De Nardis, L.; Di Benedetto, M.G. Frequentist inference for WiFi fingerprinting 3D indoor positioning. In Proceedings of the IEEE International Conference on Communication Workshops, London, UK, 8–12 June 2015; pp. 809–814.
47. Dudani, S.A. The Distance-Weighted K-Nearest- Neighbor Rule. *IEEE Trans. Syst. Man Cybern.* **1976**, *SMC-6*, 325–327.
48. Caso, G.; De Nardis, L.; Lemic, F.; Handziski, V.; Wolisz, A.; Di Benedetto, M.G. ViFi: Virtual Fingerprinting WiFi-based Indoor Positioning via Multi-Wall Multi-Floor Propagation Model. *IEEE Trans. Mob. Comput.* **2019**, doi:10.1109/tmc.2019.2908865.

49. Caso, G.; De Nardis, L. Virtual and Oriented WiFi Fingerprinting Indoor Positioning based on Multi-Wall Multi-Floor Propagation Models. *Mob. Netw. Appl.* **2017**, *22*, 825–833.
50. Rusli, M.E.; Ali, M.; Jamil, N.; Din, M.M. An Improved Indoor Positioning Algorithm Based on RSSI-Trilateration Technique for Internet of Things (IOT). In Proceedings of the 2016 International Conference on Computer and Communication Engineering (ICCCE), Kuala Lumpur, Malaysia, 26–27 July 2016; pp. 72–77. doi:10.1109/ICCCE.2016.28.