Dear author,

Please note that changes made in the online proofing system will be added to the article before publication but are not reflected in this PDF.

We also ask that this file not be used for submitting corrections.
Location-based IoT applications on campus: The IoTtalk approach

Yi-Bing Lin, Yun-Wei Lin *, Chung-Yun Hsiao, Shie-Yuan Wang

College of Computer Science, National Chiao Tung University, Taiwan

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Internet of Things
IoTtalk
Location tracking
LoRA
LTE
Mobility management

ABSTRACT

The National Chiao Tung University is deploying several location-based IoT applications on campus based on an IoT device management platform called IoTtalk. The applications include dog tracking, emergency buttons, and indoor/outdoor environment conditions monitoring (PM2.5, temperature, CO2, and so on). Some of the IoT devices for these applications have simple hardware structures to save energy, and therefore are not equipped with the positioning sensors (e.g., GPS or iBeacon). To support mobility management for these simple IoT devices, we develop a location finding mechanism in IoTtalk. By introducing the locator device in IoTtalk, we can effectively support mobility management for simple IoT devices that do not have location positioning capability. We describe how to develop the device applications to accommodate the location update feature, and show how to configure the location finding mechanism through the IoTtalk GUI. Then we conduct analytic analysis and simulation to investigate the accuracy of location tracking and power consumption for the dog tracking application.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Over the past 20 years, outdoor location-based applications have been developed based on the cellular telecommunications technologies [1]. Examples include advanced metering infrastructure, fleet management, and smart bus applications. These applications are deployed in either GPRS or LTE based broadband services. In the recent year, wireless technologies have been deployed for Internet of Things (IoT) applications with low data rate transmission, e.g., LoRA [2] and NB-IoT [3]. In National Chiao Tung University (NCTU), we have developed several location-based IoT applications, including temperature/PM2.5 detections, emergency button, and dog tracking. These applications utilize the GPS coordinates or other positioning information to identify the locations of the IoT devices, and are created by using the IoTtalk platform [4].

IoTtalk is a lightweight version of EasyConnect [5], which can flexibly manage IoT devices and their connections. In IoTtalk, an IoT device is characterized by its functionalities or “features”. A feature is a specific input or output “capability” of the IoT device. For example, an IoT device with the temperature sensor has the input device feature (IDF) called “temperature”. A wearable glasses with the optical head-mounted display has the output device feature (ODF) called “display”. Clearly, a sensor of an IoT device is an IDF, and an actuator is an ODF.

An IoT device may be connected to the network (i.e., Internet) directly or indirectly through a smartphone. If so, a corresponding software program called “network application” can be easily developed (or even automatically created) and executed in IoTtalk, which receives/sends the messages from/to the IoT device. When the values of the IDF's are updated, the

* Corresponding author.

E-mail addresses: linyb@cs.nctu.edu.tw (Y.-B. Lin), jyneda@gmail.com (Y.-W. Lin), phoebe.cyhsiao@gmail.com (C.-Y. Hsiao), shieyuan@cs.nctu.edu.tw (S.-Y. Wang).

http://dx.doi.org/10.1016/j.pmcj.2017.06.022
1574-1192/ © 2017 Elsevier B.V. All rights reserved.
An IoT device will inform the network application to take some action, and after the action, the network application may send the result to the ODF of an IoT device. In this way, the IoT devices “talk to” each other through their features. For example, we may connect the CO2 IDF of an input device to the motor ODF of an electric fan such that when a high CO2 level is detected, the fan motor runs faster. IoTtalk serves as an IoT device feature management system to provide reusable network applications for new sensors. One of the most important device features is “Location” that plays a major role for location-based applications.

In this paper, we describe how mobility management is achieved in the location-based applications using IoTtalk and LoRA deployed in the NCTU campus. LoRAWAN is a Low Power Wide Area Network (LPWAN) specification intended for wireless battery operated IoT devices, which provides seamless interoperability among IoT devices without the need of complex local installations and gives back the freedom to the user, developer, businesses enabling the roll out of IoTs [2].

Based on four location-based IoT applications deployed in the NCTU campus, this paper proposes the location finding mechanism in IoTtalk to provide the location information of the IoT devices that are not equipped with the positioning sensors. We describe how to develop the device application that accommodates the location update feature, and show how to configure the location finding mechanism through the IoTtalk GUI. We also propose analytic analysis and simulation to investigate the accuracy of location tracking and power consumption for the dog tracking application. Our study provides guidelines to balance against tracking accuracy and energy consumption.

The paper is organized as follows. Section 2 gives an overview to IoTtalk. Section 3 describes the location-based applications being deployed in NCTU. Section 4 conducts analytic analysis and simulation to investigate the accuracy of location tracking and power consumption for the dog tracking application. Also, Appendix elaborates on the device application installed in the IoT device to support mobility management of IoTtalk.

2. The IoTtalk architecture

This section introduces IoTtalk by reiterating the description in [5]. Fig. 2.1 illustrates the IoTtalk architecture that consists of two domains. The device domain (Fig. 2.1(b)) includes the IoT devices (Fig. 2.1(c)) and optionally a mobile device (smartphone; Fig. 2.1(d)). In the network domain (Fig. 2.1(a)), the IoTtalk server (Fig. 2.1(e)) implements the modularized network applications for the IoT devices. There are two scenarios for the device applications (DA) in the device domain. In the external DA scenario, an IoT device connects to the IoTtalk server indirectly through a smartphone. The wireless communications technology between the IoT device and the smartphone can be Bluetooth, WiFi, Zigbee, etc. The mobile device accesses IoTtalk through WiFi, 3G, LTE, NB-IoT, or LoRa using HTTP-based REST application programming interfaces (APIs) [6]. Alternatively, in the internal DA scenario (Fig. 2.1(f)), the IoT device connects to the IoTtalk server directly without the relay of a mobile device.

The IoTtalk network domain consists of four modules. The Creation, Configuration and Management module (CCM; Fig. 2.1(1)) systematically categorizes the features of the IoT devices, manages the mapping functions to automatically configure connectivity of IDFs and ODFs, and stores all related information in the Database module (DB; i.e., a SQL database; see Fig. 2.1(2)). The DB stores the data structures for device features and the corresponding applications that are modularized program segments for arbitrary IDF/ODF combinations. The DB is accessed by other IoTtalk modules through Object Relational Mapping (ORM) API. The Execution and Communication module (EC; Fig. 2.1(3)) consists of two submodules. The Communication submodule (CSM; Fig. 2.1(4)) retrieves/delivers the IDF/ODF information from/to the IoT devices. The IoT devices transparently communicate with each other as well as the network applications through the EC. The Execution submodule (ESM; Fig. 2.1(6)) executes network applications for the connected IDFs and ODFs. The Graphical User Interface (GUI; Fig. 2.1(7)) provides a friendly user interface to quickly establish the connections and meaningful interactions among the IoT devices. Through REST-based APIs, the GUI instructs the CCM to create or set up device features, mapping functions and connection configurations. Through UNIX signals, the CCM instructs the EC to carry out interactions between the linked IDFs and ODFs in the preset configuration of IoT devices.

The IoT device application (IDA; Fig. 2.1(8)) implements the sensor and/or actuator software to be executed in the IoT device hardware, and provides the driver for wireless communication toward the network/mobile device through the device.
application (DA; Fig. 2.1(5)). In the external DA scenario, the DA is installed in a mobile device (e.g., a smartphone running on Android, iOS, Linux or Windows). In the internal DA scenario, both IDA and the DA are installed in the IoT device. The DA consists of two software components. The device application to the network (DAN; Fig. 2.1(9)) communicates with the EC for IDA registration and data exchange. When an IoT device attaches to IoTtalk, the DAN initiates the registration procedure to inform the CCM of this attachment. After the registration, a corresponding network application assigned to the IoT device is executed by the EC. The device application to IoT device (DAI; Fig. 2.1(10)) communicates with the IoT device following the message format specified by the IDA (typically a string delivered through Bluetooth). DAI also implements sensor fusion algorithms (such as Kalman Filter and Extended Kalman Filter (EKF), sensor data storage and sensor power management. Details of DA will be elaborated in Appendix. Note that IoTtalk is very flexible and can be easily modified to be accommodated by oneM2M [7], AllJoyn [8], OpenMTC [9], WuKong [10] or any proprietary protocols. Such accommodation is achieved by porting the CSM on top of these protocols, and other modules of IoTtalk need not be modified.

3. The location-based IoT applications in NCTU

This section describes four location-based IoT applications being deployed in the NCTU campus. The first application tracks the dogs hanging around the campus. The dog-tracking device (Fig. 3.1(a)) uses a GPS receiver to identify the current location of the dog and a LoRA module to transmit the location information to the LoRA gateway (the base station). The LoRA IoT module and outdoor gateway are products of Gemtek, and the tracking application is maintained by Careyc [11]. This application utilizes the LoRA technology and the internal DA scenario to connect to IoTtalk. Fig. 3.1(b) is the photo of the first dog being tracked. The battery of the tracking device can last for three days. Therefore the NCTU dog club is responsible for charging the battery for every three days. About 30 dogs will be tracked on campus.

The second application is emergency button where a student can press a button called Qmote in case of emergency, and the alert signal is sent to the campus police through IoTtalk. Qmote (Fig. 3.2(a)) is a single-button remote control that can be hand-carried and easily accessed [12]. Qmote is connected to a mobile phone with Bluetooth Low Energy (BLE), i.e., it is connected to IoTtalk through the external DA scenario, which utilizes the GPS sensor of the smartphone to provide the location information when the Qmote's button is pressed. Therefore, Qmote is considered as a mobile IoT device connecting to the IoTtalk server through a mobile phone.
The third application monitors PM2.5 conditions where the IoT devices are stationary PM2.5 sensors. The wireless technology is LoRA, and the distances between the LoRA gateway and the sensor devices are within the range of 2000 m. Fig. 3.3(a) illustrates the primary deployment with three PM2.5 sensor devices (A–K), and two deployed LoRA gateways (Fig. 3.3(b)). An outdoor PM2.5 sensor device (Fig. 3.3(c)) uses a Sharp's G3 PMS3003 PM2.5 sensor, which is placed inside the “umbrella tube” to protect it from rain while allowing air to flow through it. We plan to deploy more than 10 PM2.5 sensor devices on campus in the near future, and the LoRA service will be offered by Careyc [11]. This application follows the internal DA scenario. Unlike applications 1 or 2 that have GPS sensors to report their locations dynamically, this application needs a “location finder” to identify the location of every PM2.5 sensor device.

The fourth application is an indoor robot called Zenbo (Fig. 3.4(a)) that connects to the Internet through Wi-Fi or LTE [13]. Zenbo can detect the locations of a building through the video camera or iBeacon. This robot carries independent sensor devices such as the indoor PM2.5/temperature/humidity device (Fig. 3.4(b)) [15] and a multi-sensor device called MorSensor [4]. The MorSensor is an IoT device where multiple sensors can be arbitrarily combined and stacked into a rectangular bar or other shapes (Fig. 3.4(c)). One or more sensors are packaged in a rectangular slice. A MorSensor device can be dynamically reconfigured with different sensors, and can quickly and conveniently accommodate new sensors through IoTtalk while MorSensor devices are still in operation (the power is not turned off). In this application, the Zenbo robot serves as the “locator” (i.e., the positioning mechanism) for the sensor devices carried by the robot.

IoTtalk provides the location finding mechanism to identify the location of an IoT device that does not equip with any positioning sensor, e.g., the outdoor PM2.5 sensor device in the third application and the indoor sensors carried by Zenbo in the fourth application. The location finding mechanism utilizes a “locator” that can be any mobile device capable of identifying the location of itself (e.g., a smartphone with a GPS receiver or Zenbo with an iBeacon receiver and/or other positioning mechanisms). The locator is an input device in IoTtalk, which has a location IDF called Location-I. For the description purpose, we use the third application as an example to illustrate the location finding mechanism. This PM2.5 system consists of several PM2.5 sensor devices distributed in different locations in the NCTU campus. These sensors are low-power, inexpensive IoT devices, which are connected to the IoT server through wireless technology such as LoRa, NB-IoT, WiFi and so on. When a sensor device reports its PM2.5 measure, its identity together with the measured value is sent...
to the PM2.5 system. Since the PM2.5 sensor devices are not equipped with any location sensors, the PM2.5 system does not know the sensor locations.

The network application of the outdoor PM2.5 system in IoTtalk includes two input devices called the locator (i.e., the GPS application in a smartphone) and the PM2.5 system, and an output device called PM2.5-map. The output device displays the PM2.5 values at different locations of the NCTU campus map. The connections of the input and the output devices can be easily achieved through the IoTtalk GUI (Fig. 2.1 (7)). In this GUI, an input device is represented by an icon placed at the left-hand side of the window. The input device icon consists of smaller icons that represent IDFs. An output device is represented by an icon placed at the right-hand side of the window, which includes ODFs. The icon in Fig. 3.5(a) represents the input device for the PM2.5 system with the IDF PM2.5-I. PM2.5-I sends the (Id, value) pair in the JSON format to the IoTtalk server, where “value” is the PM2.5 value measured by the sensor device with the identification “Id”. The locator device icon (Fig. 3.5(b)) has one IDF Location-I. Location-I outputs the GPS coordinates in the JSON format. In Fig. 3.5(c) the output device PM2.5-map is a Google map application that has two ODFs PM2.5-O and Location-O.

To obtain the GPS coordinates of the PM2.5 sensor devices, the PM2.5 system operates in two phases. In the initialization phase, Location-I of the locator is connected to Location-O of the PM2.5-map (see Join 1 in Fig. 3.5 (1)), and PM2.5-I of the PM2.5 system is connected to PM2.5-O of the map (see Join 2 in Fig. 3.5 (2)) by dragging the line segments linked between the device feature icons in the IoTtalk GUI. The PM2.5-map device maintains a location mapping table that consists of the (identification, location) entries, where the identifications are obtained from the PM2.5 system and the location coordinates are obtained from the locator. The locator periodically sends its GPS coordinates to the map though Join 2. When the user installs a PM2.5 sensor device and turns it on, this sensor device will immediately send the first (Id, value) pair to the PM2.5 system, and the system will forward this (Id, value) pair to the PM2.5-map through Join 1. The PM2.5-map checks if the identification “Id” can be found in any entry of its location mapping table. If not, an (identification, location) entry is created in the mapping table, where the identification component is set to “Id” obtained from PM2.5-O and the location component is set to the GPS coordinate obtained from Location-O. Since the user carries the locator when she/he installs the PM2.5 sensor device, it is reasonable to say that the sensor resides at the same location as the locator. Also, when the entry of the location mapping table is created, the location of the sensor in NCTU’s campus map will be highlighted. Therefore, the user can confirm the sensor’s location in the map during installation.

At the end of the initialization phase, all PM2.5 sensor devices are paired with their locations in the table of the map. In the operation phase, when a PM2.5 sensor device sends the (Id, value) pair to the IoTtalk server, the data is forwarded to the PM2.5-map device through Join 2 (Fig. 3.5 (2)). The PM2.5-map device checks if “Id” is found in an entry of the location mapping table. If so, the GPS coordinate of the entry is retrieved, and is used to display “value” in the campus map.

The configuration of the fourth application in IoTtalk GUI is similar to that for the third application, except that the locator device (i.e., Zenbo) is always connected to the map device, and the map device does not need to maintain a location mapping table. Every time the temperature or the PM2.5 sensors report new values, the map device uses the location information sent from Zenbo to identify the location in the building map.

Application 1 uses a device equipped with GPS sensor to track a dog, and does not need an extra locator device provided by IoTtalk. In this case, IoTtalk can be used to create a location management system for this application. We show that some well-known tracking features can be easily implemented with IoTtalk. For example, when the dog has moved out of an area, or when it is near a person who is afraid of dogs, IoTtalk can automatically generate warning messages. An advantage of IoT-talk-based location management system is that we can easily add the dogs, other moving objects (persons) and the alert (warning) methods to the system by configuring devices in the IoTtalk GUI (Fig. 3.6) with little or no programming efforts. In this figure, the movement of a dog (Dog 1) is monitored and when it is near a person (Person 1), he/she is alerted through a short message.

The tracking devices (Fig. 3.1(a)) represented by the “Dog1” icon (Fig. 3.6(a)) and the “Person1” icon (Fig. 3.6(b)) in the IoTtalk GUI are the same as the locator device in Fig. 3.5(b). Therefore, the network application software for these devices can be reused in IoTtalk. The Dog-map device is a Google map application like the PM2.5-map in Fig. 3.5(c) with the following exceptions. The Dog-map is represented by one input and one output device icons. The output device (Fig. 3.6(c)) consists of

Please cite this article in press as: Y.-B. Lin et al., Location-based IoT applications on campus: The IoTtalk approach, Pervasive and Mobile Computing (2017), http://dx.doi.org/10.1016/j.pmcj.2017.06.022.
Fig. 3.6. The dog management system.

Fig. 3.7. The Python function for detecting nearby dog.

\[ d = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}. \]  

(1)
4. Determining the frequency of tracking

In the dog tracking application, the tracking device periodically reports its location to the LoRA gateway. For \( i \geq 1 \), suppose that the tracking device sends the \( i \)th location report at time \( t_i \) and the next location report at time \( t_{i+1} \), respectively. Let \( \tau_i = t_{i+1} - t_i \). If \( \tau_i \) is short, the tracking device will consume too much battery power. If \( \tau_i \) is long, the location information shown in the map may be obsolete for a long time before it is update, and during this time period, the dog may have moved far away. This section investigates the tracking performance in terms of the power consumption and tracking accuracy. Let \( L_i \) be the location of the dog at \( t_i \) and \( d_i \) is the distance between \( L_i \) and \( L_{i+1} \) computed using Eq. (1). Let \( d = E[d_i] \) be the expected distance traveled by the dog in \( \tau_i \). Let \( \tau = E[\tau_i] \). Then the movement speed \( \lambda \) of the dog is expressed as

\[
\lambda = \frac{E[d_i]}{E[\tau_i]} = \frac{d}{\tau} \tag{2}
\]

where \( d \) and \( \tau \) are measured in an observation period. In the dog tracking application, if the \( \tau \) value is selected such that at every moment, the discrepancy between the location of the dog shown in the map and its actual location is less than a threshold \( d_f \), then tracking of the mechanism is considered accurate within the error \( d_i \). Selection of the period \( \tau \) is typically affected by the mobility \( \lambda \) of the dog. A small \( \tau \) should be selected if \( \lambda \) is large.

There are two types of tracking mechanisms: fixed and adaptive. The adaptive mechanism will be addressed in a separate paper. In this paper, we focus on the fixed \( \tau \) mechanism. In this widely used mechanism, \( \tau \) is a predetermined fixed value \([1–3]\). That is, \( \tau = E[\tau_i] = \tau \). Although this mechanism is very popular, it does not guarantee 100% accuracy of tracking. In other words, the mechanism is accurate with the probability \( \epsilon(d_f) = \text{Pr}[d_i \leq d_f] \). The energy consumption can be measured by a frequency index \( \theta = \frac{1}{\tau} \), where \( \theta \) is the number of location reports sent from the tracking device to the LoRA gateway during the observation period \( T \). It is clear that \( \theta \) is large if \( \lambda \) is large.

The fixed \( \tau \) mechanism is modeled as follows. We assume that \( d_i \) has a Gamma density function \( f_d(d_i) \) expressed as

\[
f_d(d_i) = \frac{\beta^\alpha d_i^{\alpha-1} e^{-\beta d_i/\alpha}}{\Gamma(\alpha)} \tag{3}
\]

where \( \alpha \) is the shape parameter and \( \beta \) is the rate parameter (or the inverse scale parameter). The mean is \( E[d_i] = \frac{\alpha}{\beta} \) and the variance is \( V[d_i] = \frac{\alpha}{\beta^2} = \frac{\beta E[d_i]^2}{\alpha} \). From Eq. (2),

\[
E[d_i] = \lambda E[\tau_i], \quad V[d_i] = \left( \frac{\lambda^2}{\alpha} \right) E[\tau_i]^2 \quad \text{and} \quad \beta = \frac{\alpha}{\lambda E[\tau_i]} \tag{4}
\]

The Gamma distribution is selected because it can be used to approximate many other distributions as well as measured data \([1]\). The fixed \( \tau \) value can be determined by using Eq. (2):

\[
E[\tau_i] = \tau = \gamma \left( \frac{d_f}{\lambda} \right) \quad \text{where} \quad 0 < \gamma \leq 1. \tag{5}
\]

The weight factor \( \gamma \) is introduced in Eq. (5) so that the tracking device reports its location earlier than the average time for the dog to reach \( d_f \). By selecting a smaller \( \gamma \), better accuracy \( \epsilon(d_f) \) is expected at the cost of more power consumption \( \theta \). Substitute (5) into (4) to yield

\[
E[d_i] = \gamma d_f, \quad V[d_i] = \left( \frac{\gamma^2}{\alpha} \right) d_f^2 \quad \text{and} \quad \beta = \frac{\alpha}{\gamma d_f}. \tag{6}
\]

In other words, the expected distance increases as \( \gamma \) and \( d_f \) increase, and its variance increases as the mean increases, and decreases as \( \alpha \) increases. Substitute Eq. (6) into Eq. (3) to yield

\[
f_d(d_i) = \frac{\left( \frac{\alpha}{\gamma d_f} \right)^\alpha d_i^{\alpha-1} e^{-\left( \frac{\alpha}{\gamma d_f} \right) d_i}}{\Gamma(\alpha)} \tag{7}
\]

From Eq. (7), the accurate tracking probability is

\[
\epsilon(d_f, \alpha, \gamma) = \int_{d_i=0}^{d_f} \frac{\left( \frac{\alpha}{\gamma d_f} \right)^\alpha d_i^{\alpha-1} e^{-\left( \frac{\alpha}{\gamma d_f} \right) d_i}}{\Gamma(\alpha)} dd_i. \tag{8}
\]

Since Eq. (6) states that \( E[d_i] = \gamma d_f \), \( \epsilon \) in Eq. (8) is independent of \( d_f \). That is,

\[
\epsilon(d_f, \alpha, \gamma) = \epsilon(\alpha, \gamma) \quad \text{if} \quad E[d_i] = \gamma d_f. \tag{9}
\]

Indeed, if \( \alpha \) is a positive integer, then Eq. (8) is expressed as

\[
\epsilon(\alpha, \gamma) = 1 - \sum_{n=0}^{\alpha-1} \left( \frac{\alpha}{\gamma} \right)^n e^{-\frac{\alpha}{\gamma}} \frac{n!}{n!}. \tag{10}
\]
For $\alpha = 1$ Eq. (10) is further simplified as

$$\epsilon (\gamma) = 1 - e^{-\frac{1}{\gamma}}.$$  

The power consumption index $\theta$ is always affected by $\lambda$, $\gamma$, and $d_T$. From Eq. (5),

$$\theta (\lambda, \gamma, d_T) = \frac{1}{\tau} = \frac{\lambda}{\gamma d_T}. \quad (11)$$

The analytic analysis Eq. (8) is validated against the Monte Carlo simulation. The discrepancies are within 1%.

In Eq. (9), $\lambda$ is determined by the dog’s behavior, which can be observed through the dog’s movement history. Based on $\lambda$, we can adjust $\gamma$ to optimize $\epsilon$ against $\theta$. Suppose that we fix both $\tau$ and $\gamma$. As we pointed out, the dog’s mobility $\lambda$ cannot be controlled, and if we changes $d_T$, then Eq. (5) does not hold. Intuitively, if $d_T \rightarrow \infty$, $\epsilon \rightarrow 1$. Practically, a small $d_T$ should be selected, and if a small $\epsilon$ is to be achieved, we should adjust both $\tau$ and $\gamma$ to satisfy Eq. (5). In the remainder of this section, we assume that $\tau$ and $\gamma$ are selected such that Eq. (5) holds.

Fig. 4.1(a) plots $\epsilon$ against $\alpha$ and $\gamma$. In this figure, the relationship $E[d_i] = \gamma d_T$ holds. From Eq. (6), if $E[d_i]$ is fixed, then the smaller the $\alpha$ value, the larger the variance $V[d_i]$ of the dog’s movement. The figure indicates that for a fixed $E[d_i]$, the accuracy $\epsilon$ increases then decreases as $\alpha$ increases (or the variance $V[d_i]$ increases). When $V[d_i]$ is small, the $d_i$ values do not significantly vary. Since $\gamma \leq 1$, it is likely that $d_i < d_T$, and large $\epsilon$ is expected. On the other hand, when $V[d_i]$ is large, the $d_i$ values significant vary. For a fixed $E[d_i]$, a large $V[d_i]$ means that there are more small $d_i < d_T$ and fewer $d_i \gg d_T$. In this case, a large $\epsilon$ is also expected.

Since we select $\tau$ such that Eq. (5) holds, Eq. (11) indicates power consumption. Fig. 4.1(b) plots the $\theta$ curves. We assume that $d_T = 50$ m. It is obvious that $\theta$ decreases as $\lambda$ decreases and $\gamma$ increases. The figure indicates that when $\gamma > 0.5$, increasing $\gamma$ only insignificantly affects $\theta$. Consider the case where $d_T = 50$ m, $\lambda = 20$ m/s, and the variance $E[d_i]$ of the dog movement is medium (e.g., $\alpha \approx 1$). To achieve the accuracy $\epsilon = 0.85$, $\theta = 0.8$ per second. In some IoT communications systems, the report frequency is limited to much lower values (e.g., 1 per second). Such systems are not appropriate for dog tracking.

5. Conclusions

This paper described several location-based IoT applications deployed in the NCTU campus. The first application tracks the dogs hanging around the campus, which provides better management of the campus dogs. The second application is emergency buttons where a student can press a button called Qmote in case of emergency, and the alert signal is sent to the campus police. This application will replace the fixed emergency poles in NCTU. The third application monitors the PM2.5 conditions where the IoT devices are stationary PM2.5 sensors. The fourth application is an indoor robot that carries several sensors and can detect the environment conditions through the routes of the building it travels. These applications using different positioning methods are implemented using the IoTtalk platform, and IoTtalk provides the same mobility management to these applications.
The IoT devices in these location-based applications may not be equipped with the positioning sensors. To resolve this issue, this paper proposes the location finding mechanism in IoTtalk to provide the location information for these IoT devices. Our study indicated that the mechanism functions well. Specifically, we described how to develop the device application that accommodate the location update feature, and showed how to configure the location finding mechanism through the IoTtalk GUI. We have conducted analytic analysis and simulation to investigate the accuracy of location tracking and power consumption for the dog tracking application. Our study provides guidelines to balance against tracking accuracy and energy consumption.

NCTU is deploying several wireless technologies including LoRa, NB-IoT, and ultra-dense LTE networks to support the location-based IoT applications, and we will evaluate the performance of the location-based applications based on these wireless technologies. In particular, the LTE network and the applications called M-Echo will be deployed with Alpha Networks Inc. [15], and IoTtalk will be enhanced with the sensor solution of Connected Automation Global Inc. [16].

Acknowledgments

The Java DA is implemented by Chang-Yen Chih (the external DA scenario) and Qing Liu (the internal DA scenario). The LoRA gateway and the IoT devices are provided by Amingo Huag of Careyc. The robot Zenbo is provided by David Lo of ASUS. The Qmote device is provided by Samson Chen of Qblink. The NCTU smart campus project is supported by Quanta (especially Ted Chang) and Chunghwa Telecom (especially Rick Wang).

Appendix. Device application for mobility and communications

To accommodate IoT devices to IoTtalk, we provide an API called the CSMapi. The developer uses the CSMapi to write the DA software to communicate with the CSM (Fig. 2.1 (4)) through message exchange. A message can be a command for IoT device control such as mobility management or the DF data (e.g., temperature or a Qmote button click). Fig. A.1 illustrates the flows for the command and data messages delivered in the device domain. Just like mobile telecommunication systems [1], we define the transmission path from the IoTtalk server to the IoT device as the downlink and the transmission path from the IoT device to the IoTtalk server as the uplink. It is clear that the IDF data are delivered through uplink, and the ODF data are delivered through downlink. Depending on the direction of command delivery, we define the downlink commands (dl_cmd) and the uplink commands (ul_cmd). As pointed out in Section 2, the IDA communicates with the DA in one of two scenarios: internal or external. In the internal DA scenario, both the DA and the IDA are co-located in the IoT device (e.g., the dog-tracking device in Fig. 3.1(a)). In the external DA scenario, the IDA and the DA are deployed in different places, e.g., the DA in a smartphone and the IDA in an IoT device (e.g., the Qmote emergency button). For the internal DA scenario (Fig. A.1(a)), the IDA and the IDA are implemented together (in the IoTtalk server; Fig. A.1 (3) and (4)). The DAN invokes the CSMapi functions (Fig. A.1 (1)) to communicate with the CSM in the IoTtalk server; Fig. A.1 (2)). The DA/IDA invokes the DAN functions to communicate with the CSM. The device domain processes the messages sent from the CSM to the IDA by using the functions pull() in the CSMapi, the DAN and the DA/IDA. The push() functions are invoked in the reverse order. For the external DA scenario (Fig. A.1(b)), the DA and the DA resides in the same location (e.g., a smartphone; Fig. A.1 (6) and (7)), and the IDA (Fig. A.1 (8)) is located at a separate IoT device (Fig. A.1 (5)).

Fig. A.2 illustrates the DA functional blocks. Four tasks (Fig. A.2(a)-(d)) in the DA are executed to communicate with the CSM (Fig. A.2(e)) through the CSMapi (Fig. A.2(f)). Each of the four tasks is divided into two parts implemented in the DAN and the DAi respectively. The DAN is responsible for delivering messages and maintaining registration status between the DA and the CSM. The DAi is responsible for message transmission and connection status maintenance between the DA and the
IDA (Fig. A.2(g) and (h)). The device application format conversion (e.g., sensor data encoding/decoding and unit conversion) is also handled by the DAI. Both the internal and the external DA scenarios are implemented with multiple threads. The main thread executes the initialization and registration task to create two threads. The first thread executes the DAN and parts of the DAI that processes the downlink messages. The second thread executes the IDA and parts of the DAI that processes the uplink messages. After creating these two threads, the main thread is terminated.

The IoTtalk protocol consists of the user and the control planes [1]. The user plane delivers measured data sent to/from the IoT devices. The control plane delivers commands between IoTtalk and the IoT devices. At the system level, IoTtalk defines four commands: registration (Fig. A.2(a)), pull (including get-location request; see Fig. A.2(b)), push (including location-update; see Fig. A.2(c)) and deregistration (Fig. A.2(d)), which are required for all DAs and are implemented in the CSMapi (Fig. A.2(f)). Specifically, the get-location command is issued from the IoTtalk server to query the IoT device’s location. The location-update command is periodically sent from the IoT device to the IoTtalk server to report the device’s location.

At the application level, IoTtalk defines two special device features Control-IDF and Control-ODF to transfer device dependent command messages between the IoTtalk server and the DA. Control-IDF/ODF has two components, i.e., a String variable specifying the command name and a JSON object [17] storing the command parameters. Three default commands are defined for the Control-IDF/ODF:

- Device feature status: to indicate the selected IDFs and ODFs for connection. This command has one parameter, i.e., a status list that indicates the on/off status of each device feature.
- Resumption: to resume (or start) data transmission of all IDFs and ODFs (except for Control-IDF/ODF). This command has no parameters.
- Suspension: to suspend data transmission of all IDFs and ODFs (except for Control-IDF/ODF). This command has no parameters.

Suppose that an IoT device has three input device features ID $F_i (1 \leq i \leq 3)$ and four output device features ODF $j (1 \leq j \leq 4)$. If the user selects the device features ID $F_1$ and ODF $3$, then IoTtalk sends the “Device feature status” command to the IDA with parameters “SET_DF_STATUS” and (“cmd_params”: [“1000010”]) to activate ID $F_1$ and ODF $3$ of the IoT device for connection. In this example, ID $F_1$ and ODF $3$ are enabled, and other IDFs/ODFs are disabled and are not used. The string “1000010” (the status list) in the second parameter indicates the status of each device feature, where the $i$th bit represents the status of ID $F_i$ $(1 \leq i \leq 3)$, and the $(3 + j)$-th bit represents the status of ODF $j$ $(1 \leq j \leq 4)$. Note that after the IoT device has registered to IoTtalk, Control-IDF/ODF are always connected until deregistration.

To support location-based applications, the mobility management commands (get-location and location-update) are implemented as Control-IDF/ODF that cannot be suspended. The temperature sensor and emergency button are implemented as ordinary IDFs that can be suspended.
A.1. The external DA scenario

In this scenario, the CSMapi and the DAN are the same as those in the internal DA scenario. The design and implementation of the DA and the IDA for the internal DA scenario is discussed in [18]. This section elaborates on the external DA scenario. The IoT Talk DA has three versions implemented in C#, Java and Python. This section describes the Java implementation. We assume that the IoT device communicates with the mobile phone through BLE. Fig. A.3 illustrates the details of the DA functional blocks for the external DA scenario, where the black rectangular boxes stand for a host or a device (e.g. CSM); a horizontal dashed line stands for an API; the gray rectangular boxes stand for Java classes; the black oval boxes stand for Java functions; and the white oval boxes stand for Java inner classes, a special Java class defined within another class. Since the IoT device is connected to the smartphone through BLE, the IDA is called BLE_IDA. We implement several Java classes including DAN (Fig. A.3(a)), DAI (Fig. A.3(b)), and BLE_IDA (Fig. A.3(c)).

Consider an IoT device with \( I \) IDFs, \( J \) ODFs, and \( K \) commands. For the description purpose, denote the \( i \)-th IDF as IDF-\( i \) (which can be, e.g., “Temperature”), the \( i \)-th ODF as ODF-\( i \) (which can be, e.g., “MapLocation”), and the \( i \)-th command as COMMAND-\( i \) (which can be, e.g., “GET_LOCATION”). Note that the response for COMMAND-\( i \) is denoted as COMMAND-\( i \)_RSP (which can be, e.g., “GET_LOCATION_RSP”). The response for a downlink (uplink) command is an uplink (downlink) command. For every IDF-\( i \), ODF-\( j \) and COMMAND-\( k \), the DA developer creates the classes with the exact names IDF-\( i \) (Fig. A.3(d)), ODF-\( j \) (Fig. A.3(e)) and COMMAND-\( k \) (Fig. A.3(f)), respectively. For example, for the Temperature IDF, the class Temperature is created. The class COMMAND-\( i \) handles both downlink and uplink commands of the \( i \)-th command. The MessageQueue class is implemented in the IDA to ensure that the messages sent to the IoT device are transmitted in order (i.e., transmit the next message only after the current response is received).

A.2. Registration and connection

The initialization and registration task consists of the following steps.

**Step 1**: The main() function (Fig. A.3(g)) is the entry point of the DAI class. This function calls BLE_IDA.init() (Fig. A.3(h)) to initialize the member variables of the BLE_IDA class.

**Step 2**: After member variable initialization, BLE_IDA.init() calls BLE_IDA.search() to scan the neighboring IoT devices to obtain the MAC address of the first scanned device.

**Steps 3–6**: BLE_IDA.init() then calls BLE_IDA.connect() (Fig. A.3(i)) to connect to the IoT device (Fig. A.3(j)). If the BLE connection is successfully established, BLE_IDA.init() returns the MAC address of the IoT device to main(). Otherwise, an empty string is returned.
Step 7: If the BLE connection is set up, `main()` calls `DAN.init()` (Fig. A.3(k)) to initialize the member variables of the DAN class. Otherwise, the DA terminates.

Step 8: If the IoT device is equipped with a user interface (e.g., the touch screen of a smartphone) to allow the user to input the CSM’s IP address, this step is skipped. Otherwise, `DAN.init()` calls `DAN.search()` to scan the CSMs of the IoTTalk servers under the same local area network (LAN). `DAN.search()` returns the IP address of the first scanned CSM. Note that many IoTTalk servers may be active and are used by different users at the same time.

Steps 9–12: The DAN registers to the CSM by invoking `DAN.register()` (Fig. A.3(l)) that calls the registration command implemented in the CSMapi. If the registration is successful, `DAN.register()` calls `start()` (Fig. A.3(m)) to communicate with the CSM. In our design, the protocol used between the DAN and the CSM is HTTP, which means that the server cannot intensively exchange messages with the DA. The DAN periodically calls the pull command implemented in the CSMapi, and drops the received message if the timestamp is equals to the previous one. The `start()` function is provided by Java threading library, which invokes `run()` to execute the pull command.

Steps 13 and 14: `DAN.register()` returns the result to `DAN.init()`. If the registration operation is successful, `DAN.init()` returns the URL of the CSM to `main()`. Otherwise, an empty String is returned.

If this task is successfully executed, the IoT device is connected to the IoTTalk server.

A.3. Pull and GET_LOCATION

During execution of the initialization and registration task, the pull task is triggered (Steps 11 and 12) if the registration is successful. The pull task consists of the following steps.

Steps 15 and 16: The DAN periodically calls the pull command implemented in the CSMapi, and checks the timestamp of received message to see if the message should be dropped. If not, the DAN checks if the message is pulled from Control-ODF. If so, the DAN handles the command message. If the command is RESUME, the DAN starts pulling the selected output device features. If the command is SUSPEND, the DAN stops pulling all output device features except for Control-ODF. If the command is SET_DF_STATUS, the DAN changes the selection flags according to the command parameters. The DAN then sends the message to the DA by calling `DAN.pull()` (Fig. A.3(n)).

Step 17: `DAN.pull()` checks if the output device feature is Control-ODF. If so, the data is unpacked to retrieve the command name and the command parameters. The command parameters are passed to the `run()` function of the corresponding COMMAND class (Fig. A.3(f)). Otherwise, the data is passed to the `pull()` function of the corresponding ODF class (Fig. A.3(e)).

Steps 18 and 19: The ODF class or the COMMAND class encodes the message into a byte array in the format that the IoT device understands, and the DA passes it to the BLE_IDA by calling `BLE_IDA.write()` (Fig. A.3(o)). `BLE_IDA.write()` puts the message into `MessageQueue` (Fig. A.3(p)) through `MessageQueue.write()`.

Steps 20–22: If `MessageQueue` is not empty, `MessageQueue.send_msg()` (Fig. A.3(q)) is called to send the message to the IoT device. `MessageQueue.send_msg()` copies the first message from `MessageQueue`, calls `writeCharacteristic()` (Fig. A.3(r)) to transmit the message to the IoT device, and then starts a timer to call itself again if the response message is not received after the certain time period (the default length is 1 s). Messages are transmitted in order, and the next message is not transmitted before the previous response has been received.

The GET_LOCATION Control-ODF is implemented in the pull task in IoTTalk following Steps 15–22.

A.4. Push and LOCATION_UPDATE

The push task is the counterpart of the pull task in the reverse direction, which consists of the following steps.

Steps 23–26: The message transmitted through BLE is received through the `onReceive()` callback (Fig. A.3(s)), and is passed to `MessageQueue.receive_msg()` (Fig. A.3(t)). `MessageQueue.receive_msg()` calls `DAN.msg_match()` (Fig. A.3(u)) to check if the message is a response of a previous message sent to the IoT device. If so, the first message is removed from `MessageQueue`. `MessageQueue.send_msg()` is invoked to send the next message, and `DAN.receive()` (Fig. A.3(v)) is called to handle the response.

Steps 27 and 28: `DAN.receive()` decodes the received message to check if it is an IDF data. If so, the message is dispatched to the corresponding IDF class (Fig. A.3(d)). Otherwise, the message is dispatched to the corresponding COMMAND class (Fig. A.3(f)).

Step 29: The IDF class or the COMMAND class decodes the message and handles it (i.e., converts the unit of the data and packs it into the JSON format). If the message is successfully handled, `DAN.push()` (Fig. A.3(w)) calls `CSMapi.push()` to push the data to the CSM. Note that the command handler may produce multiple messages, for example, the SET_DF_STATUS command may be decomposed into multiple messages to be sent to the IoT device. In this case, the received multiple responses will be consolidated to form a SET_DF_STATUS_RSP command.

The LOCATION_UPDATE Control-IDF is implemented in the push task in IoTTalk following Steps 23–29.
A.5. Deregistration and disconnection

The deregistration task is triggered by the user, and is OS dependent (e.g. the onDestroy() callback in Android or the shutdown hook in a standard Java environment). The deregistration task consists of the following steps.

Steps 30 and 31: DAI.deregister() (Fig. A.3(y)) calls BLE_IDA.disconnect() (Fig. A.3(x)) to disconnect the IoT device.
Steps 32–36: DAI.deregister() calls DAN.deregister() (Fig. A.3(z)) that stops communicating with the CSM, and invokes CSMapi.deregistration() to disconnect from the CSM.

After the deregistration task is successfully executed, the IoT device is disconnected from the IoTtalk server.
A.6. The device dictionary at the EC

After the IDA has received the location update command or obtained the sensor data from the IoT device, the DAN sends these data to the EC. The EC stores the sensor data by the device dictionary, a Python data type called dictionary, which is represented by an unordered set of key-value pairs [19]. A value sent from an IoT device (e.g., temperature = 31 °C) is stored in the EC as a real number (e.g., 31) and is indexed by a key (e.g., temperature). Fig. A.4 shows the dictionary structure for an input IoT device such as MorSensor and an output device such as a bulb, and the Python code is listed in Fig. A.5.

The key of the device dictionary (Fig. A.4(a)) is the MAC address of the IoT device (e.g. C860008BD249; see Line 1 in Fig. A.5). Under the device dictionary, a second level dictionary called “profile” (see Lines 2–9) maintains the device name d_name (e.g., My Sensor; see Line 3), the device model dm_name (e.g., MorSensor; see Line 4), the device feature list df_list (e.g., [“Acceleration”, “Temperature”]; see Lines 5–7), the power consumption indication pwr (e.g., 100 mWh; see Line 8) and the location loc (see Line 9). The location of the IoT device can be tracked if it is equipped with GPS or indoor iBeacon applications (e.g. fb0b57a2-8228-44). The device dictionary also stores the IDF data sent from the input device features or the ODF data sent from the network application. These data are stored as samples in the IDF/ODF items (see Lines 10–21). The IDFs/ODFs of an IoT device are specified in its df_list. In Fig. A.5, the df_list is [“Acceleration”, “Temperature”], and two of the IDF/ODF items are created with keys “Acceleration” (Line 10) and “Temperature” (Line 16) under the device dictionary. The IDF items store the data samples which are sent from the device features of the IoT device in the descending timestamp order (e.g. [“2015-06-22 19:30:40”, [0, 0, 9.8], ...]; see Lines 11–15 and Lines 17–21). This example shows 5 samples of the Acceleration and the Temperature IDF, respectively. Complete sample traces can be collected and stored in a log file.

With the “loc” entry in the profile dictionary, IoTtalk can conduct mobility management similar to mobile telecommunications network [1] with specific consideration for the IoT characteristics as pointed out in Section 2.

References