ARCHIVIO ISTITUZIONALE DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Designing a Hybrid Push-Pull Architecture for Mobile Crowdsensing using the Web of Things

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Sciullo, L., Montori, F., Zyrianoff, I., Gigli, L., Tinti, D., Di Felice, M. (2023). Designing a Hybrid Push-Pull Architecture for Mobile Crowdsensing using the Web of Things. 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA : IEEE COMPUTER SOC [10.1109/smartcomp58114.2023.00081].

Availability:
This version is available at: https://hdl.handle.net/11585/959688 since: 2024-02-20

Published:

DOI: http://doi.org/10.1109/smartcomp58114.2023.00081

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

18 November 2026

Designing a Hybrid Push-Pull Architecture for Mobile Crowdsensing using the Web of Things

Luca Sciullo*†, Federico Montori*†, Ivan Zyrianoff*†, Lorenzo Gigli*†, Davide Tinti*, Marco Di Felice*†,
* *Department of Computer Science and Engineering, University of Bologna, Italy*
† *Advanced Research Center on Electronic Systems "Ercole De Castro", University of Bologna, Italy*
Emails: {luca.sciullo, federico.montori, ivandimitry.ribeiro, lorenzo.gigli, marco.difelice3}@unibo.it, davide.tinti3@studio.unibo.it

***Abstract*—Mobile crowdsensing (MCS) is an emerging paradigm that leverages the pervasive presence of mobile devices to collect and analyze data from the environment. However, the choice of a push- or pull-based architecture for MCS can result in a loss of flexibility and limitations for the creators of the campaigns (crowdsourcers). To address this issue, we propose a hybrid push-pull architecture for MCS campaigns that leverages the W3C Web of Things (WoT) to standardize the interfaces and interactions of devices through well-consolidated Web technologies. Our architecture supports different kinds of campaigns at the same time and balances the amount of data with its quality, enabling geo-localization of MCS workers to further improve data quality. Furthermore, we present the design and implementation of a WoT-enabled Android application for MCS. We evaluate our proposal through simulations in a vehicular scenario based on a real dataset, showing that the hybrid architecture provides greater flexibility to crowdsourcers, supporting simultaneously the push and pull paradigms.**

***Index Terms*—Mobile Crowdsensing, W3C Web of Things, Smartphones, Android, simulations**

I. Introduction

The modern era is marked by the widespread use of mobile phones, which are ubiquitous and equipped with various sensors, such as a camera, microphone, GPS, accelerometer, digital compass, light sensor, and Bluetooth used as a proximity sensor. According to the IDC Worldwide Smartphone forecast [1], shipments of smartphones are expected to reach 1.53 billion units by 2026, increasing, even more, the presence of such devices in our lives. This represents a significant opportunity for leveraging the vast number of mobile devices available to create a geographically distributed sensing infrastructure. Mobile crowdsensing (MCS) [1] is a paradigm that leverages the participation of a large number of mobile devices, such as smartphones, to collect and analyze data about the physical world. In particular, entities interested in collecting such data, defined as *crowdsourcers*, initiate a data collection campaign to which mobile phone users, defined as *workers* or *participants*, can subscribe and participate by collecting sensor data and uploading it to the cloud. MCS can be used in a wide range of different applications, including environmental monitoring, healthcare, and traffic management. However, the success of MCS is heavily reliant on user acceptance, and as such, rewards are often offered to incentivize users to share their data. These rewards could be in the form of discounts, benefits, or credit [2], but gamification of the data collection [3] has been investigated too. Considering all these aspects, several different architectures for MCS have been proposed in the last years, which can be generally categorized as either *push-based* or *pull-based* architectures [4].

In push-based architectures, workers are explicitly requested to complete a task, which allows for more precise control over who contributes to the campaign and ensures a higher quality of collected data. On the other hand, pull-based architectures require workers to intentionally retrieve the list of available campaigns and decide freely to contribute without any specific request from the server. This approach is particularly suitable for phenomena of common interest that require multiple perspectives to be accurately described and can tolerate potential lower-quality samples given the higher number of participants. However, relying solely on a single push- or pull-based architecture may result in a loss of flexibility, not only in the architecture itself but also in the communication technologies between the entities involved in the architecture. A push-based interaction requires that the mobile device is reachable and able to receive requests, while a pull-based interaction requires the mobile device to simply send data to an external entity. Once the architecture is deployed, crowdsourcers are limited to a single way of interaction with the workers, which limits their possibilities to dynamically adapt the campaigns to different scenarios.

This paper presents a novel hybrid architecture that combines both push- and pull-based approaches, resulting in increased flexibility for campaign creators. To achieve this, we leverage the W3C Web of Things, which aims to standardize device interfaces and interactions using established Web technologies. By enabling devices to expose their capabilities through different Web protocols such as HTTP and MQTT, our architecture allows for interaction with both push- and pull-based approaches at the same time. More in detail, the paper presents three main contributions:

- We propose a hybrid architecture both for push- and pull-based MCS. The same architecture can support different campaigns, granting full flexibility for the crowdsourcers. Furthermore, the architecture enables the geo-localization of workers, improving further the quality of data while reducing the amount.

[1] https://www.idc.com

- We present the design and implementation of a W3C WoT-compliant Android application for collecting, exposing, and sending data to the campaigns. The application follows the specifications of the W3C WoT servient [5], as better described in Section IV.
- We validate our architecture by leveraging a traffic control campaign in a simulated scenario based on traffic conditions on top of sensor data collected in a real traffic dataset.

The rest of this paper is structured as follows: in Section II we provide a brief introduction of the background in MCS and technologies used for the geo-localization, and a review of the W3C WoT standard and its components. Section III introduces the architecture, while its implementation is described in Section IV. Section V introduces the validation through simulations, while conclusions and future works are discussed in Section VI.

II. Background and Related Work

In this section, we aim to introduce the current state of knowledge in the field, which is fundamental for the proposed architecture. Namely, we address the IoT-based architectures for MCS, the W3C WoT standard, and LA-MQTT – an extension to the standard MQTT to support location awareness. Further, we highlight our advances in each of the mentioned topics.

A. Architectures for Mobile Crowdsensing

MCS has gained momentum in the last decades, driven by the widespread adoption of mobile devices (e.g., smartphones, IoT devices). However, this paradigm imposes a new breed of demands on system architectures. They need to handle multiple mobile workers and an intense data flow while managing payments and privacy issues [6].

In the scope of this article, we focus on the task scheduling features of MCS architectures, which is the process of allocating tasks for workers. There are two well-defined strategies for scheduling, namely *push-* and *pull-based*, also known as *pro-active* and *reactive* [6], respectively. In push-based architectures, the worker receives the request to perform a task assigned by the crowdsourcer. On the other hand, in pull-based architecture, the workers proactively decide when and to which tasks they want to contribute.

The authors of [7] propose a push-based auction framework for MCS where the platform acts as an auctioneer to recruit workers for a sensing task. The proposed framework involves workers submitting privacy-preserving versions of their data and the platform selecting a subset of workers based on their sensing capabilities. Another push-based architecture was proposed in [8], where workers can trade their location privacy for a higher chance of being selected to perform tasks. Since individuals perceive their location privacy differently, each worker can decide how much information they wish to disclose. In [9], the authors tackled the issue of task assignment in push-based MCS. First, they create an assignment graph for each available worker, considering their spatiotemporal mobility and the dangling task list; then, they utilize an optimization algorithm to match workers and tasks. Finally, in [10] the authors propose a novel push-based framework encompassing the spatiotemporal correlation of the sensed data to reduce the number of allocated tasks while ensuring data quality.

Among pull-based architectures, we highlight the Cost-Aware Compressing Sensing (CACS) [11], in which each sensing device calculates the probability of transmitting the sensed data independently, taking into account factors such as the device energy, the transmission cost, and the requirements for real-time data. Another notable architecture was proposed in [12], in which the authors leverage pull-based MCS to implement a reverse auction strategy that determines the size of the incentive paid to workers. The existing literature has explored various pull- and push-based MCS approaches, however, our approach differentiates from the current state-of-the-art since no architecture supports both methods simultaneously.

B. Web of Things

The Web of Things (WoT) paradigm aims to counter the fragmentation of IoT environments by leveraging and extending well-known Web standards. The most notorious efforts on the specification and standardization of the WoT paradigm are developed by the World Wide Web Consortium (W3C) WoT Interest Group, which established a detailed standard on the implementation level [5]. It defines a standard protocol-agnostic interface that enables easy interaction and integration of heterogeneous IoT devices to platforms and applications [13].

A Web Thing (WT) is an abstraction of a virtual or physical entity that exposes a Thing Description (TD), which is the formal information model and common representation of a WT [14]. The TD describes the WT metadata, capabilities, properties, and interactions described in a machine - and human-understandable structure, i.e., JSON or JSON-LD. The interaction with a given WT is implemented through protocol bindings, which define the mapping of the possible device interactions to different network protocols.

The execution environment of a WT is called *servient*. The servient can interact with remote WTs by consuming their TDs and can host and expose WTs, in fact performing the role of client and server at the same time. Although servient description is implementation-agnostic, the *de facto* standard implementation is `node-wot`[2], which is written in Typescript and maintained by the W3C Working Group. There are other servient implementations, as in Python [15] and Java [16]. However, that are no servient implementations that can natively be executed in Android.

Another critical WoT building block envisaged in the WoT architecture [5] is the Thing Description Directory (TDD) [17]. The TDD is a WT indexer that enables search capabilities on the registered WTs metadata (i.e., their TDs). Further, it allows registering, deleting, and updating WTs dynamically.

[2]https://github.com/eclipse/thingweb.node-wot

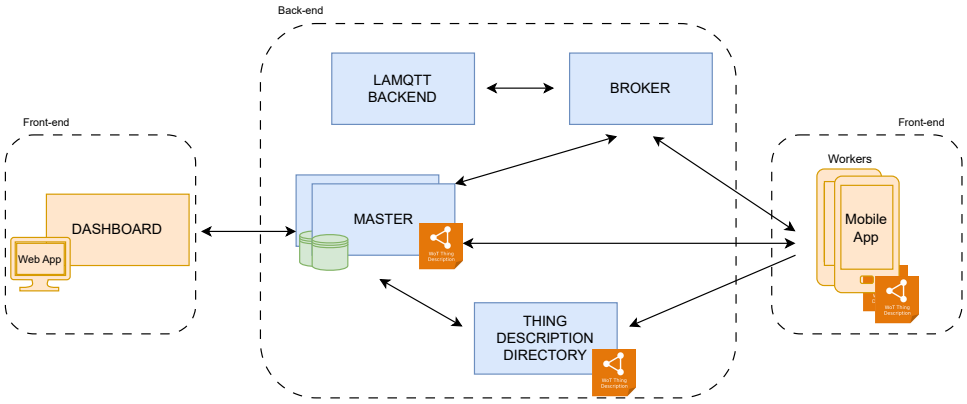


Fig. 1. The micro-services oriented architecture proposed.

A known shortcoming of the W3C WoT standard is the lack of out-of-the-box conversion methods to dissonant interfaces to its ecosystem. Implementation efforts are often needed to integrate third-party Web services or other standard interfaces (e.g., NGSI-LD) into the WoT. Recent advances filled that gap, providing seamless integration of RESTful Web services [18] and NGSI-based interfaces [19] to the W3C WoT ecosystem. The current work can leverage those new developments by offering – through the W3C WoT – plug-and-play capabilities with the mentioned technologies. Our work represents meaningful progress to the WoT standard. As far as we know, no MCS architecture incorporates the WoT paradigm to handle device heterogeneity.

C. LA-MQTT

MQTT is a lightweight publish-subscribe protocol that is commonly used in IoT-based scenarios. However, the standard MQTT protocol does not support location awareness, a key feature for MCS or any application scenario that supports Location-Based Services, since MCS campaigns are typically restricted to a well-defined geographic area. `LA-MQTT` is an extension to standard MQTT supporting spatial-aware communications while considering privacy implications by adding location-awareness [20]. In `LA-MQTT`, consumers are only notified with relevant data in terms of location and topic. Those features are implemented mainly through a back-end application pluggable in standard MQTT brokers. Although the subscribers are only notified on relevant data, the `LA-MQTT` imposes some additional message exchanges to keep track of the position of the devices; the mobile IoT nodes must constantly update the broker, i.e., the location's `LA-MQTT` back-end.

III. Software Architecture

As described in Section II-A, the existing architectures proposed for MCS are either *push-* or *pull-based*, with the consequence of limited flexibility. This prevents the creation of campaigns that can support both approaches, thus limiting the possibilities of crowdsourcers to adapt the campaigns to a different kind of scenario.

The architecture proposed is micro-services oriented, where every macro-functionality has been integrated into a single component and can be deployed independently. As shown in Figure 1, the architecture consists of two front-end services and four back-end components; in the first case, the *Dashboard module (DM)* can be considered as the GUI of the system, through which crowdsourcers can create new campaigns and visualize the data collected, while the Mobile Application (MA) is responsible for collecting data for the workers. The back-end layer is composed of a *Master Module (MM)*, an *MQTT broker (MQTT)*, a *Thing Description Directory (TDD)*, and the *LA-MQTT (LA)* module [20]. More in detail, through the *DM* a crowdsourcer can create a new campaign, customizing its settings, such as the number of samples requested, the geographic boundaries for data, the duration of the campaign, the typology of sensors required, and whether the campaign is push or pull-based. Furthermore, every campaign requires a title and a brief description to be advertised to the workers. The DM also lets crowdsourcers manage the campaigns, visualizing the data collected or stopping an active campaign. Every request made through the DM is then forwarded to the MM to be handled and stored in a local storage. Every new campaign is turned and deployed as a Web Thing (WT), hence exposing all its interactions as affordances to be consumed by a servient: this automatically enables multiple protocols for the same interaction (like HTTP or MQTT), that can be used for both the push and pull-based approaches. Once the WTs have been produced and deployed, the MM advertises them to the TDD, so the workers can discover and join them. The TDD works also as a registry for the workers, where they can register themselves in order to be selected directly from the DM for a push-based campaign. The registration to the TDD is automatically done by the MA of the workers, which also acts as GUI through which the workers can interact with the rest of the architecture. In particular, a worker can visualize the list of active campaigns, select the ones it is interested in joining, and start to collect data for a campaign according to its settings. In the case of a push-based campaign, crowdsourcers will ask for data on demand from the workers. This is possible through the usage of the external MQTT broker, which enables bidi-

rectional communication between crowdsourcers and workers. Additionally, if the crowdsourcer requires geo-localized data, it can ask for data in a specific zone thanks to the use of the LA module, as better depicted in Figure 3. On the contrary, in the case of a pull-based campaign, the worker selects one of three different approaches for sending the data: (i) *automatic*, where data is sent automatically matching the campaign preferences; (ii) *customized*, where data is sent according to a customized interval; and (iii) *manual*, where workers send data manually. The MA lets also workers specify what sensors are enabled for the crowdsensing and the default interval between each new data sent for the *customized* approach.

A. Architecture Interactions

The architecture interactions are detailed in Figure 2 and Figure 3, respectively for the pull- and push-based approaches. In the first case, asynchronously workers get registered to the TDD. At one point, a crowdsourcer creates a new campaign through the dashboard; this request is forwarded to the MM which creates a WT with all the information of the campaign and publishes it on the TDD. A worker can now manually get the list of available campaigns and decide which ones to join. Periodically, the worker collects data that is sent directly to the MM, which stores it. Eventually, once the campaign is over, the crowdsourcer can retrieve the campaign data from the MM.

In the second case, we need to distinguish the flow of operations for the classic push-based approach (blue dashed arrows) and for the localized one (green dotted arrows), which makes use of LA-MQTT. Like in the pull-based flow, we can suppose that a worker registers on the TDD *(0)*, but in order to enable the geo-localization features of the architecture, from this moment onwards it periodically sends its position to the MQTT broker *(3b)*, which forwards it to the LA *(4b)* in order to be processed. When a crowdsourcer publishes a new campaign *(1)* to the TDD through the DM (*2*), for the classical approach it also retrieves the list of workers from the TDD *(8a, 9a, 10a*), thus it can decide which ones to query. The worker can ask for the list of active campaigns from the TDD *(5, 6)* and decide which ones to join *(7)*, then it waits for the data requests from the MM.

When the crowdsourcer starts the campaign (*11*), the MM forwards the request for new data to a specific worker passing through to the broker in case of the classical approach (*12a*), or asking for new geo-localized data. For the latter purpose, the request is again forwarded to the broker *(13b)*, but the broker retrieves the list of geo-localized workers from the LA (*14b, 15b)*. In both cases, the broker publishes the data request *(16)* and the worker sends back the data to the MM through the broker *(18, 19)* after collecting it (*17*). In the end, asynchronously and after the campaign has been completed, the crowdsourcer gets the data back from the MM (*20*) and visualizes it through the DM *(21)*.

We highlight that in both the previous cases no reward system has been considered since it is out of the scope of this paper, but any kind of reward can be integrated taking advantage of the flexibility of the architecture presented.

The proposed architecture has been designed to grant a strong decoupling of the services as well as the possibility to replicate and spread them depending on the needs. This means that no specific constraint has to be taken into account in the deploying stage, hence the architecture can be considered general-purpose and can be used in several different scenarios. For example, let us can consider a smart city scenario, where the municipality provides a shared TDD where every available worker can register itself, offering its data to anyone who is interested. In this case, a single TDD can be deployed for the workers' registration, while multiple different crowdsourcers could host their private MM that interacts with the shared TDD. This approach would have the benefit of conveying all the workers into a single source, but at the same time granting the privacy and security needed for a private company interested in a crowdsensing campaign since the interactions and data are handled by different MMs.

On the contrary, let us imagine a smart city scenario where the municipality wants to collect data from distinct categories of users, for instance distinguishing between normal and expert users. This can be easily achieved through the deployment of multiple TDDs, one for each category, and a single MM for handling all the data flows. In this way, workers can only join the campaigns that were designed for the category they belong to, while the crowdsourcer can easily manage several campaigns that can have different settings. We remark that the way in which a user is assigned to a specific category can be done in several ways, both online and offline, but the details are out of the scope of this paper. The *MQTT* and *LA* modules are used for supporting the bi-directional interactions and can be deployed/replicated independently of the rest of the architecture logic.

IV. Implementation

We here illustrate the implementation details of the architecture modules. The DM is a front-end web application written in Angular[3] leveraging the the MapBox[4] library to enable geospatial data visualization and analysis. The MM [5] is a Node.js application written in TypeScript that has a twofold task: (i) it interacts with the other WoT-enabled components of the architecture and it exposes every campaign as a single WT; (ii) it saves and retrieves data from a local storage. For the first task, it executes a *WoT servient* that has been written using the mentioned `node-wot` framework . For the second one, since the local storage is a MongoDB instance, it uses the official MongoDB client library for interacting with the database. Listing 1 shows how the campaign has been mapped into a MongoDB collection. We remark that the settings of a campaign can be easily customized through the DM. The TDD contains the TDs of the registered workers and of the campaigns. For this module, we used an already existing

[3]https://angular.io

[4]https://mapbox.com

[5]https://github.com/Daveeeeed/crowdsensing-wot-proxy

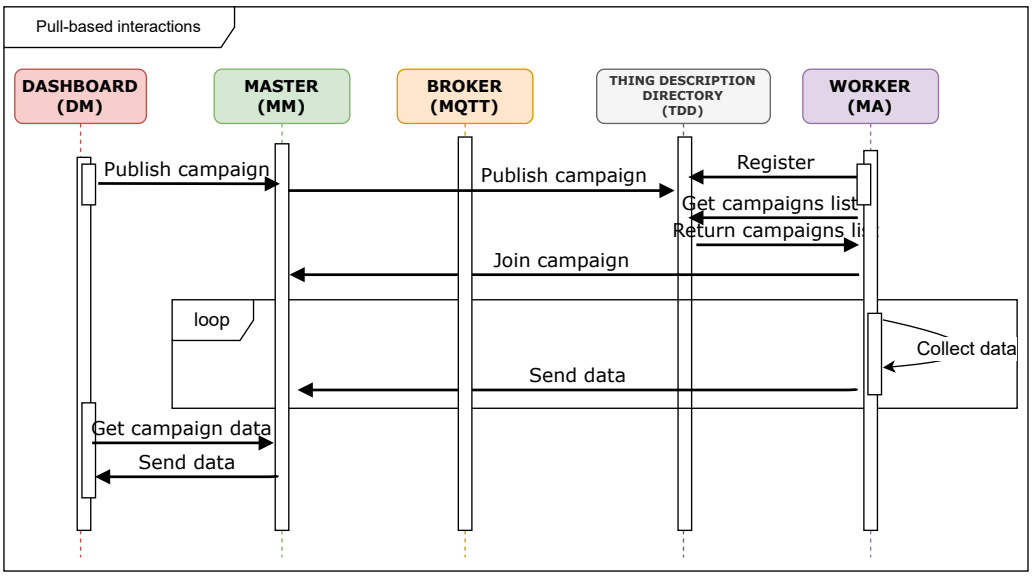


Fig. 2. The interactions for the pull-based approach.

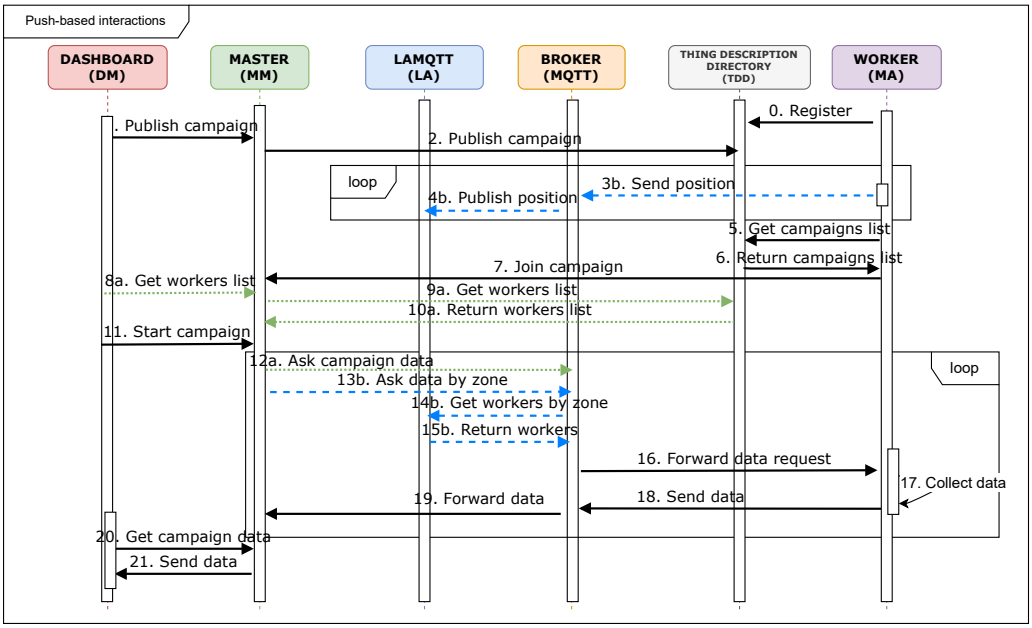


Fig. 3. The interactions for the push-based approach.

implementation written in Go of TinyIoT [6], built following the W3C specifications. The MQTT is a Mosquitto [7] broker, while the LA is a Typescript application that can be executed as a background process either on the same host of the MQTT broker or on a separate host, as detailed in [20]. Finally, the MA is an Android application, described in Section IV-A.

```
{
    "_id": ObjectID,
    "title": String,
    "description": String,
    "organization": String,
    "ideal_submission_interval": Integer,
    "points": Integer,
    "pull_enabled": Boolean,
    "push_auto_enabled": Boolean,
    "push_input_enabled": Boolean,
    "submission_required": Integer,
    "type": Enum<SubmissionType>,
    "area": {
        "center_lat": Double,
        "center_lng": Double,
        "radius": Double,
        "type": Enum<AreaType>
    }
}
```

Listing 1. The MongoDB schema used for storing a campaign.

[6]https://github.com/TinyIoT/thing-directory
[7]https://mosquitto.org/

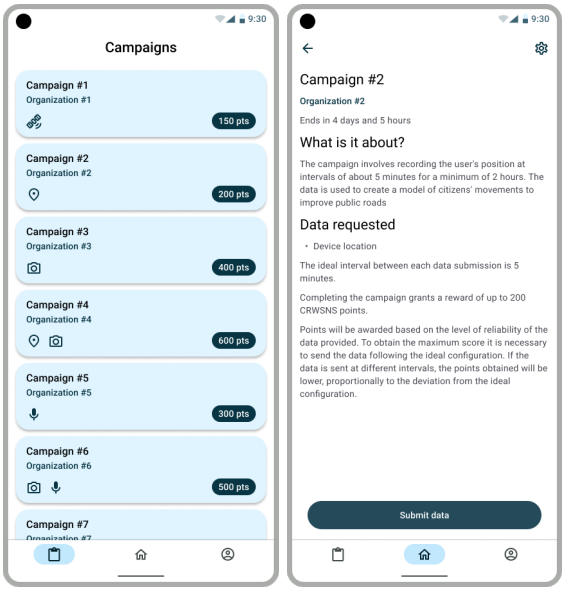
A. Mobile Application and Android Servient

This section presents the implementation of the MCS application for the Android operating system, along with the details that make the application a WoT servient. Since there are no other implementations for Android, we took inspiration from the aforementioned *SANE* servient [16]. Our implementationis meticulously designed to enhance the features of mobile devices, ensuring the following functional requirements: (i) Expose through a TD the device's internal sensors, including the location; (ii) modify the TD metadata by setting its exposed sensors and related visibility; (iii) Import a TD through a file, URL or Discovery mechanisms; (iv) Consume a WT and interact with it.

Four binding templates have been created in this implementation: HTTP, WebSocket, CoAP, and MQTT (including the LA-MQTT support). These templates are independent packages of the application, enabling straightforward addition or removal. The system uses third-party libraries for these built-in protocols: HTTP is powered by the Spark[8] library, which provides tools for starting an HTTP web server with several configuration options. For CoAP, we choose Californium[9], while the WebSocket support is part of Netty[10], an asynchronous event-driven network application framework that is highly performance-oriented and provides a scalable and maintainable implementation. Finally, MQTT leverages the Paho Android Service[11] and connects directly with an already created broker which is not part of the Servient or the device. We strongly adapted the existing implementation to encompass also LA-MQTT scenarios. The application is composed of a client and a server part. Although both of them can be run simultaneously, we decided to instantiate them separately at the application level. This architectural choice is based on a limitation that we face on the protocol side: when a protocol configuration changes, it is necessary to restart the servient. Therefore, by separating the logic of the client and the server, it will only be necessary to restart one of the two components.

The client component does not need to manually set up configurations for the protocols since all the required metadata are already contained in the TD. After starting the servient, if some consumed WTs are available (campaign WTs), the client loads them, as shown in Figure 4(a). Now it is possible to click on the row corresponding to a consumed WT and interact with it through a dedicated detail page (Figure 4(b)). The server can be run in "server-only" mode; once executing, it manages the device sensors through a dedicated service that exposes their values into the WT properties. The user can configure which sensors to expose and which ones to keep private, thus controlling his privacy.

[8]https://github.com/perwendel/spark
[9]https://github.com/eclipse-californium/californium
[10]https://github.com/netty/netty
[11]https://github.com/eclipse/paho.mqtt.android



(a) (b)

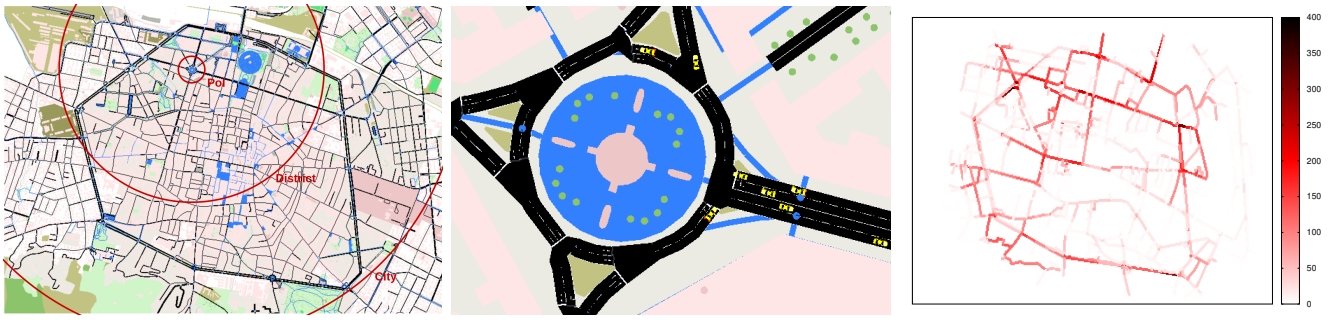
Fig. 4. List of the active campaigns and the detail of a campaign in the Android App.

V. Performance Evaluation

In this section, we aim to validate our architecture by leveraging an example campaign that wants to assess traffic conditions on top of sensor data collected by a number of workers. Specifically, the evaluation takes place through traffic simulations over the city center of Bologna, Italy. Since realistic traffic conditions are of utmost importance in order to conduct rigorous simulated evaluations, we made use of the Bologna Ringway dataset [21]. The dataset provides two hours of real traffic data, which has been collected through the installation of spires in correspondence with strategic nodes of the road network.

The dataset has been loaded as a SUMO [22] configuration and controlled via its TraCI interface using Python scripts. The city center rendered via the SUMO GUI is shown in Figure 5(a). Over 22.000 vehicles are moving through the city center of Bologna from 12:00 AM to 2:00 PM and follow their path from their spawn point to their ending point. By participating in the "traffic conditions" MCS campaign, each vehicle reports a number of sensor readings that altogether are expected to be representative of the campaign object (e.g. GPS, accelerometer, gyroscope, noise intensity in dB). All these sensor readings, which are assumed to be sent over by the MA to the MM within a single message, are hereafter defined as "observation".

According to our architecture, our campaign could potentially be conducted as push-based or pull-based, based on various conditions. The evaluation in this section aims to show the structural and quantitative differences deriving from the adoption of one or another and discusses under which conditions the push-based mode is more convenient than the pull-based one and vice versa. In the pull-based mode, we assume that each participating worker periodically sends an



(a) SUMO GUI render of the simulated scenario with the three zones of interest.

(b) Closeup of the center of all zones of interest.

(c) Heatmap of the traffic intensity in the city center.

Fig. 5. The simulation environment

observation to the MM, enclosed in a single HTTP message. In our simulation, we set the update frequency to 20s.

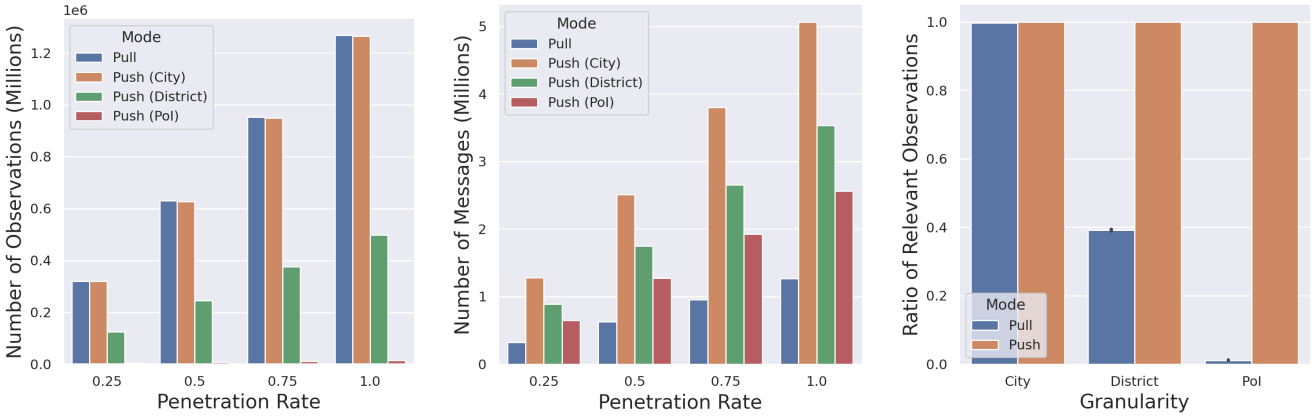
For the push-based mode, as outlined in Section III, the MM has to explicitly target a number of workers to execute the task on demand, on top of contextual information possessed by the MM itself that would designate certain workers as suitable. In our use case, we imagined such a context to be given by the location of the worker, as a crowdsourcer might be more interested in getting traffic data from a location that is particularly interesting (e.g., a trafficked junction, a spot where there has been an accident or road work), for which data would be crucial and, probably, more rewarded. For this reason, we chose to evaluate here the usage of LA-MQTT for the push-based scenario, where the crowdsourcer sets three zones of interest with different granularity: a city-wide area (radius 2.000m), a district-wide area (radius 1.000m), and a PoI-wide area (radius 100m). The data flow takes place as described in Section III, with the workers updating the MQTT broker with their position every 20s and the MM requesting an update to each worker within the zone of interest every 20s, to keep the comparison with its pull-based counterpart meaningful. All three zones, shown in Figure 5(a), are circles, centered at a roundabout for which a closeup view is shown in Figure 5(b) during a running simulation. Figure 5(c) shows the vehicle distribution over the city center – each point is normalized by the number of vehicles passing through it on an average of 10 minutes –, highlighting how the traffic is mainly concentrated along the ring road and the main arteries of the city, with some junctions getting quite crowded.

Our results aim to evaluate three fundamental metrics:

1) The total number of **observations** sent from the workers to the MM. The higher the number, the higher the coverage, however, an excessive number of observations may cause the crowdsourcer to reward too many contributors, as well as observations being not meaningful for the considered phenomenon (see point 3).
2) The total number of **messages**, which may give an idea of the network load caused by the adoption of a pull-based mode, which has a minimal data exchange protocol, as opposed to a push-based mode, which needs more interaction.
3) The number of **relevant observations** received by the MM, which gives insights on the overall data quality of the campaign. In this case, we consider relevant only observations taken within the zone of interest.

Results assessing these metrics are shown in Figure 6 and are all evaluated on top of the *penetration rate* of the application, which is the ratio of vehicle drivers that participate in the campaign – the total number is 22.210. Figure 6(a) shows the number of observations using the two modes. The number of observations sent using the pull-based mode does not change on top of the zone of interest, thus we showed a single value for that mode. It is evident how the number of observations drops significantly for the push-based mode for a smaller zone of interest. Figure 6(b) shows the overall number of messages, which evidently validates how the pull-based scenario causes a lower network load than its push-based counterpart, regardless of the size of the zone of interest. This is expected, as the LA-MQTT protocol needs additional messages and checks that occur even if the actual observation is not sent. Finally, Figure 6(c) shows the performance of the pull-based mode, in terms of relevant observations, on top of the size of the zone of interest. It is evident how a small zone of interest yields poor performance, as pull-based workers would send their observations even if they are not within the designed area, causing the majority of the data to be low quality.

These results show that different scenarios can be suitable for pull- or push-based campaigns on top of certain parameters, in this case, the size of the zone of interest: push-based mode yields better performances for a small zone, whereas pull-based mode achieves nearly the same performance of push-based mode for large areas with much lesser overhead. This evaluation does not consider many other aspects – e.g., a pull-based mode can be driven by the client application, which may perform additional checks (i.e., being inside the zone of interest); however, our architecture cannot enforce this. In fact, given its standard WoT interface, we can envision multiple entities developing a dedicated client application, which falls outside the architecture responsibility. Nevertheless, this evaluation shows that it might be more valuable to use either pull- or push-based mode depending on the evaluated scenario. Thus, it is crucial for a system to support both modes.



(a) Number of observations sent by all workers. (b) Number of messages sent by all workers. (c) Success ratio of both modes.

Fig. 6. Simulation results.

VI. Conclusions and Future Works

In this paper, we proposed a hybrid push- and pull-based architecture for mobile crowdsensing campaigns, taking advantage of the W3C Web of Things to ensure flexibility and standardization. Our architecture can support different campaigns and enable geo-localization of workers, balancing the amount and quality of collected data. Future research could focus on (i) the support of hybrid campaigns, optimizing the trade-off between the amount and quality of data in the same campaign, (ii) privacy preservation, integrating all the features of the LA-MQTT library, and (iii) an extension of the evaluation by considering real large-scale scenarios, like a smart city.

References

[1] J. A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," 2006.

[2] L. G. Jaimes, I. J. Vergara-Laurens, and A. Raij, "A survey of incentive techniques for mobile crowd sensing," *IEEE Internet of Things journal*, vol. 2, no. 5, pp. 370–380, 2015.

[3] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?–a literature review of empirical studies on gamification," in *2014 47th Hawaii international conference on system sciences*. Ieee, 2014, pp. 3025–3034.

[4] M. Louta, K. Mpanti, G. Karetsos, and T. Lagkas, "Mobile crowd sensing architectural frameworks: A comprehensive survey," in *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 2016, pp. 1–7.

[5] W3C Working Group. (2021) WoT Reference Architecture (W3C Recommendation 9 April 2020). [Online]. Available: http://www.w3.org/TR/wot-architecture/

[6] A. Capponi, C. Fiandrino, B. Kantarci, L. Foschini, D. Kliazovich, and P. Bouvry, "A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2419–2465, 2019.

[7] W. Gong, B. Zhang, and C. Li, "Task assignment in mobile crowdsensing: Present and future directions," *IEEE Network*, vol. 32, no. 4, pp. 100–107, 2018.

[8] W. Jin, M. Xiao, M. Li, and L. Guo, "If you do not care about it, sell it: Trading location privacy in mobile crowd sensing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1045–1053.

[9] L. Wang, Z. Yu, K. Wu, D. Yang, E. Wang, T. Wang, Y. Mei, and B. Guo, "Towards robust task assignment in mobile crowdsensing systems," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2022.

[10] L. Wang, D. Zhang, Y. Wang, C. Chen, X. Han, and A. M'hamed, "Sparse mobile crowdsensing: challenges and opportunities," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 161–167, 2016.

[11] L. Xu, X. Hao, N. D. Lane, X. Liu, and T. Moscibroda, "Cost-aware compressive sensing for networked sensing systems," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, ser. IPSN '15. ACM, 2015, p. 130–141.

[12] T. Matsuda, T. Inada, and S. Ishihara, "Communication method using cellular and d2d communication for reverse auction-based mobile crowdsensing," *Applied Sciences*, vol. 12, no. 22, p. 11753, 2022.

[13] L. Sciullo, L. Gigli, F. Montori, A. Trotta, and M. D. Felice, "A survey on the web of things," *IEEE Access*, vol. 10, pp. 47 570–47 596, 2022.

[14] W3C Working Group. (2023) Web of Things (WoT) Thing Description 1.1. [Online]. Available: https://www.w3.org/TR/wot-thing-description11/

[15] A. G. Mangas and F. J. S. Alonso, "Wotpy: A framework for web of things applications," *Computer Communications*, vol. 147, pp. 235–251, 2019.

[16] H. Bornholdt, D. Jost, P. Kisters, M. Rottleuthner, D. Bade, W. Lamersdorf, T. C. Schmidt, and M. Fischer, "Sane: Smart networks for urban citizen participation," in *2019 26th International Conference on Telecommunications (ICT)*, 2019, pp. 496–500.

[17] W3C Working Group. (2023) Web of Things (WoT) Discovery (W3C Candidate Recommendation Snapshot 19 January 2023). [Online]. Available: https://www.w3.org/TR/wot-discovery/

[18] I. Zyrianoff, L. Gigli, F. Montori, C. Kamienski, and M. D. Felice, "Two-way integration of service-oriented systems-of-systems with the web of things," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, 2021, pp. 1–6.

[19] I. Zyrianoff, A. Heideker, L. Sciullo, C. Kamienski, and M. Di Felice, "Interoperability in open iot platforms: Wot-fiware comparison and integration," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2021, pp. 169–174.

[20] F. Montori, L. Gigli, L. Sciullo, and M. D. Felice, "La-mqtt: Location-aware publish-subscribe communications for the internet of things," *ACM Transactions on Internet of Things*, vol. 3, no. 3, pp. 1–28, 2022.

[21] L. Bedogni, M. Gramaglia, A. Vesco, M. Fiore, J. Härri, and F. Ferrero, "The bologna ringway dataset: Improving road network conversion in sumo and validating urban mobility via navigation services," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 12, pp. 5464–5476, 2015.

[22] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.