A STACK4THINGS-BASED PLATFORM FOR MOBILE CROWDSENSING SERVICES

S. Distefano

Kazan Federal University
Higher Institute for
Information Technology and Information Systems
Kazan, Russia
Email: sdistefano@kpfu.ru

*A. Puliafito, G. Merlino, F. Longo, D. Bruneo**

Università di Messina
Department of Engineering
Messina, Italy
Email: {apuliafito,gmerlino,
flongo,dbruneo}@unime.it

ABSTRACT

As mobiles grow pervasive in people's lives and expand their reach, Mobile CrowdSensing (MCS) and similar paradigms are going to play an ever more prominent role. There is a pressing need then to ease developers and service providers in embracing the opportunity, and that means offering a platform for such efforts. This in turn means providing a solid foundational architecture with abstractions and sound layering for MCS application designs to be mapped over it. This should base on a flexible infrastructure able to provide resources to MCS applications according to their requirements, hopefully on-demand. A service-oriented/Cloud model can perfectly fill this gap. This paper is a first step in this direction, proposing to adopt Stack4Things (S4T), an OpenStack-based platform for managing sensing and IoT nodes, for runtime customization of resources and their functions to support MCS services and applications. This implies developing and extending the S4T platform further to the specific requirements coming from off-the-shelf, e.g., Android-based, mobiles, as well as describing an example S4T-powered MCS application, Pothole Detection Mapping, to highlight the role of the platform.

***Keywords*—** Mobile crowdsensing, Cloud, IoT, OpenStack, Android.

1. INTRODUCTION AND MOTIVATIONS

Mobile CrowdSensing (MCS) comprises by definition a category of applications where individuals carrying sensor-hosting embedded computers (e.g. smartphones) get collectively engaged in information gathering and sharing efforts to analyze and georeference events which may be interesting for individuals and communities alike. One of the main advantages of MCS is the possibility to conduct sample collection, data mining, etc., without accounting for the corresponding experiments in advance, just leveraging natural daily life patterns arising from human activities as they happen and leave behind breadcrumbs in form of samplings ready to be collected.

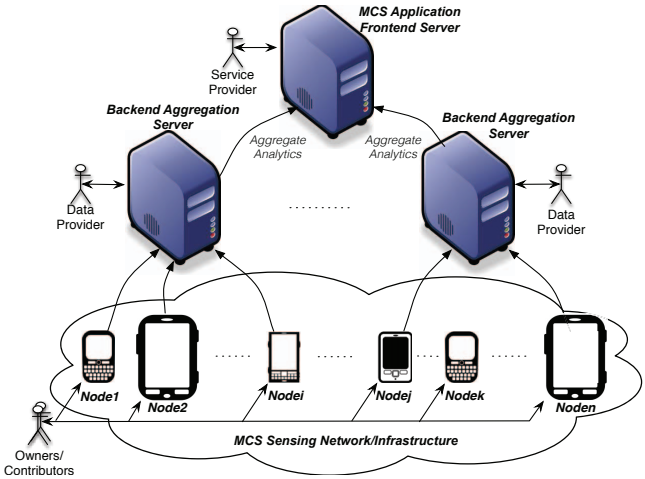


Figure 1. The MCS reference scenario.

Typical MCS applications mainly implement a client-server interaction pattern where a *service provider* offers MCS-based services to *end users*, leveraging *contributors* willingness to provide their physical (sensing) resources. Data are therefore collected and processed by (backend and frontend) servers to carry out aggregate analytics and feed back relevant results to end users. Starting from the lowest level, through heuristics and algorithm design, local analytics may provide a category of functions, among which simple ones are interpolation, extrapolation and outlier filtering, which may enhance a standard MCS application as shown in Figure 1. A few drawbacks of such siloed pattern limiting the potential of the underlying paradigm are: i) unoptimized runtime, as multiple applications would execute on the same nodes without taking into account such configuration, possibly duplicating sensing or processing activities on resource-constrained devices, thus also limiting scalability of the platform; ii) necessity of enrolling, collecting and managing resources, i.e., lack of a coordinated sens-

*This research effort was supported by the EU 7th Framework Programme under Grant Agreement n. 610802 for the "CloudWave" project and by the EU Horizon 2020 Research and Innovation Program under the Grant Agreement n. 644048 for the "BEACON" project, and has been carried out in the framework of the CINI Smart Cities National Lab.

ing resource management. In particular this last point is crucial, as some mechanisms and facilities for dealing with sensing resource management are of strategic importance to enable the actual potential of the MCS paradigm. Furthermore, specific patterns able to support the MCS paradigm by, for example, implementing coordinated, self-managed cooperation mechanisms among nodes could boost MCS to a new-hot trend in the IoT scenario, as a feasible solution for things management and related applications and services development.

This paper moves towards this direction, proposing Stack4Things [1] (S4T), a framework for the management of distributed smart objects in the IoT context through a service oriented, on-demand, cloud provisioning model, for the management of the MCS sensing infrastructure. This goes further beyond current IoT-Cloud trends, mixing the two paradigms for the management of (mainly data) resources in the cyber-physical space, towards a brand new, crowd-driven approach to IoT and related app development. Specifically, starting from the S4T infrastructure-oriented framework, the architecture is adapted and extended in the following to support MCS, mainly providing node enrolment, customization, networking and control facilities in a service-oriented/Cloud fashion. Then an example of an MCS application taking advantage of the functionalities exposed by the platform is discussed.

2. MCS AND IOT PARADIGMS

MCS as a paradigm embraces many approaches to crowd-sourcing sensor data, including both participatory and opportunistic sensing. On one hand participatory sensing [2] may be defined as any crowd-sourced sensing activity where each member of the crowd is actively involved, giving feedback when asked or otherwise tagging measurements on a voluntary basis. This is to be contrasted to an opportunistic perspective [3], where sensing is essentially unmanned: MCS would tap into mobile devices just because people carry those around in their pocket all day long anyway, and may just be involved once with a fire-and-forget experience, leaving then all crowd-sensing activities to unassisted background processes.

As devices are carried around by individuals, owners may eventually be in the (data feeding) loop. Their mobility and situational awareness may be leveraged, in an opportunistic and participatory fashion respectively, to support the collection of finer grained information and semantically tagged data. For MCS applications to succeed, there have to be appropriate incentive mechanisms to recruit, engage and retain human participants. In this sense, a centralized credit system, assigning and managing credits and rewards, is usually adopted as incentive mechanism in a participatory strategy. On the other hand, in opportunistic scenarios, the gamification approach is usually adopted, building up a credit collection race among contributors to incentive their participation.

As a more subtle differentiation of the two approaches, we may consider how they diverge in reference to the actors benefiting from the crowd-sourcing. It would follow that participatory may be considered any MCS, and services that may derive from it, when a community, or the public at large, is the one entity taking mostly (and primarily) advantage, e.g., [4, 5, 6]. Whereas opportunistic would be MCS where any outcome would eminently center around single individuals. The latter is usually linked to measurements of individual phenomena, i.e., where samples are to be traced back to individuals producing them, usually featuring other individuals as a way to augment processing on otherwise purely personal trails. Conversely, community sensing revolves around naturally anonymized aggregation of data.

Table 1. Taxonomy table of MCS applications.

MCS **categorized (by)**	**Approach**	
	Participatory	*Opportunistic*
Owner involvement	Active, human-assisted sensing / tagging	Background, unmanned data collection
User benefit	Public interest	Individual utility
Fruition modality	Pull / non-contextual	Push / contextual
Interaction model	Centralized (client-server)	Distributed (mesh)
Incentive mechanism	Credit systems (bank)	Credit collection race

While still focused on the end-user perspective, another aspect to be considered is how information produced by an MCS system is to be consumed, or made relevant to the situation under which fruition would occur. A typically proactive, participatory pattern for users may consist in merely consulting an MCS-derived knowledge base, thus leveraging information as-is, i.e., non-contextually and in a pull fashion. Conversely an opportunistic fruition mode would be based around push-based notifications, depending on certain inferred metrics on the (dynamic) environment surrounding the user, thus contextual in nature. Context itself may be exploited to dynamically allocate sensing tasks to the best subset of participants [7], or other metrics may be combined and evaluated to rank participants for such kind of allocation, e.g., measuring credible interactions among participants [8]. Moreover, also in terms of interactions, at least first-time enrollment requires input on the side of owner, employing a client-server model. Yet even opportunistic schemes, featuring distributed behavior and cooperative strategies, may be considered, dependent on the underlying topology, as is the case for mesh-like ones in device-dense environments.

A synthesis of the approaches and categories of MCS applications is presented in Table 1, in particular with reference to the multifaceted definition both kind of approaches embed, where each row represents a degree of freedom with respect to this dichotomy. This way, a wide range of possibilities for MCS application paradigms, from pure participatory to wholly opportunistic ones, may be identified, including also hybrid solutions horizontally spanning one or more axes.

MCS can be also considered as an Internet of Things (IoT)-related paradigm. Indeed, the whole IoT research community agrees on the notion that things are to be interconnected over some potentially global network (possibly, but not exclusively the Internet), to be exploited for whichever scenario, and in pa rticular by specific applications and services. One distinguishing feature is the autonomous, chatty nature of

such mesh. As now happens with applications made up of distributed components overlaid on the network, interacting by means of well-established protocols and paradigms, e.g. RESTful services, also things should network themselves and their corresponding agents, with no (direct) operator intervention (think M2M). Under such premise, MCS may just become a pattern under the IoT umbrella term, i.e. a specialization of the platform that an IoT would represent for sensing-related, mobility-enabled, crowd-sourced use cases. This perspective should be particularly appreciated in light of opportunistic developments, as attaching semantic description to things, endowed with distributed, event-triggered logic, may really provide chances for nodes to discover each other services and weave an even more powerful abstraction with respect to the Web of Things.

Although IoT can be mainly associated with the participatory pattern, some work on opportunistic IoT and sensing environment is available in literature. For example, in [9] an opportunistic IoT framework is proposed, mainly extending opportunistic networking towards participatory sensing, enabling opportunistic information sharing among things to also support mobile social networking. Similarly, opportunistic mobile networking is the topic of [10], mainly focusing on low level data forwarding issues through a framework able to support and optimize opportunistic sensing. The best way to exploit the seemingly limitless IoT potential for sensing at a global scale is by coupling opportunistic and participatory application patterns for mobile crowds with the infrastructure underneath. The proposed solution is aimed at horizontally enabling *things* for massive development and deployment of any sensing task and based on a novel application of Cloud computing technologies to IoT, as described in the following.

As we pointed out already, not only should we optimize any MCS despite shaky grounds, but we aim to turn most shortcomings to our advantage. More specifically, uneven global connectivity when coupled with dense, mesh-like topologies lends itself naturally to be addressed by means of routing protocols as those designed for MANETs. This leads to avoiding otherwise likely network congestion and servers' overload while also making room for any kind of crowd-local, opportunistic and cooperative behavior.

In particular when compared to the building blocks of MANETs par excellence such as, e.g., WSNs, nowadays consumer-class mobile devices feature plenty of computing, storage and communication capabilities, making these platforms many orders of magnitude more advanced than mote-like ones, also in terms of sensing, considering that mobiles are usually multi-modal by default, thus enabling a wider range of applications. Moreover, the impact of massive deployments in the field can never be overestimated, as these devices are available by the billions, always-on and mostly online, a consistent slice of people's daily activities and recurring habits. Leveraging such huge populations means building potentially instant-on large-scale applications while at the same time lowering expenses and time-to-market, also avoiding altogether the long setup times and high upfront costs involved in case of ad-hoc sensing infrastructure, when not absolutely required.

Dynamic behavior and chances for data reuse are also differentiating factors in evaluating MCS against WSN-like configurations. When dealing with MCS, the composition of device populations, as well as the kind and quality of data produced, where quality may be expressed both in terms of latency and precision, may change in time due to mobility patterns, power requirements and communication subsystems, including owner-mandated local preferences. In traditional WSNs the populations (and the kind of data produced) are usually known a priori, thus managing quality and designing according to requirements is more straightforward. Moreover, with regard to MCS, support for multiple concurrent applications is feasible, albeit subject to careful planning and design stages. When it comes to sensor networks instead, deployments are typically geared for a specific application, thus repurposing or resource sharing are rarely accounted for.

Still casting challenging constraints into opportunities, evolutions in time of crowds' shapes and composition may lead to fast dissemination of information, aiding in the pursuit of (global) optimization objectives. Moreover we care about actual MCS usefulness, for contributors themselves too, thus any solution can't be considered absolutely all-round if lacking built-in feedback and assisted guidance. Even with regard to data itself, both in terms of raw format and application requirements, sensor readings may not be suitable for direct consumption. In this sense specific local analytics may be performed on-board, to (pre)process raw data, in order to obtain intermediate results ready for transmission and further processing.

Even not taking into account the scalability issues any kind of backend may be subject to when dealing with huge raw data uploads, on-board resources in mobile devices are on the rise and already remarkable in many cases, thus it seems natural to tap into such potential to shift in part the burden of computations toward the edge. Moreover, a renown tradeoff in conventional mote-class WSNs lies in leveraging (local) computation to save on the amount of energy and bandwidth, as needed to transmit preprocessed data in comparison to raw readings. Last but not least, delay-sensitive applications may benefit from such savings, if the overhead of local processing is negligible when compared to transmission-induced latencies.

3. A SERVICE-ORIENTED MCS INFRASTRUCTURE

The Stack4Things framework [1, 11], also referred to as S4T, is our effort to pursue a new approach for the management of smart objects and things in the IoT scenario, following an on-demand, service oriented provisioning model. This shifts the IoT paradigm towards the Cloud one, merging benefits of both: on the one hand providing control and management capabilities to IoT (sensing and actuation) resources, on the other enriching the Cloud paradigm with pervasive I/O capabilities to directly interact with the environment. Altogether

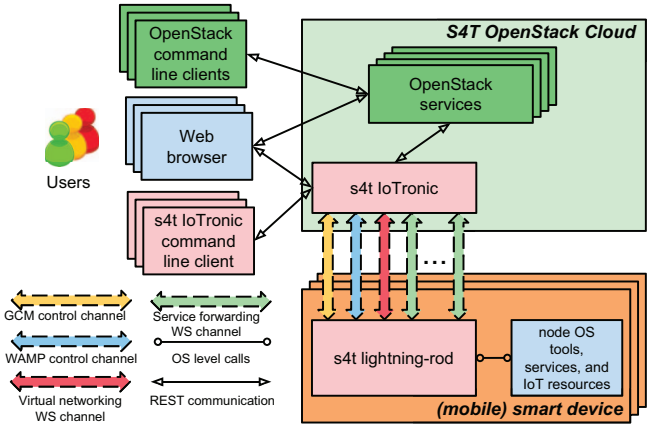


Figure 2. Stack4Things overall architecture for MCS.

these ideas and challenges have been framed into a novel utility paradigm for IoT [12], mainly highlighting, from a functional perspective, its focus, i.e., to establish a sensing-Cloud.

Key aspects of SAaaS paradigms are the i) device-centric philosophy [13] of providing actual (sensing and actuation) devices or things, even if virtualized, to higher levels (users, app developers or providers) and the ii) unified management and control of all the underlying resources. This implies to provide, on the one hand, mechanisms for customize and contextualize the resources, pushing intelligence to the extreme edge by injecting code on them and, on the other, a control surface able to manage these resources and provide them as a "utility", in a Cloud-oriented model. S4T design has thus been tackled by extending certain *OpenStack* subsystems, a well-known and widely adopted Cloud management framework, to smoothly integrate and leverage as much existing functionalities as possible.

Figure 2 shows the S4T overall architecture for MCS, derived from the one defined in [1] by specializing its modules in the MCS domain, in particular considering mobile devices. It highlights interactions and communication facilities between end users, the Cloud and mobile nodes hosting sensing (and, sometimes, even actuation) subsystems.

On the MCS node side, the *S4T lightning-rod* runs under the device-native (typically SDK-enabled) environment available for developers and interacts with a (subset of) OS services and tools available on the device for which authorizations have been granted, as well as with (physical or even virtual) sensing and actuation resources, through UNIX-style filesystem-based abstractions of the underlying interfaces, either GPIO for embedded boards or, typically, (SDK-specific) API-mediated for mobiles. As well, it connects to the Cloud and anchors the node to the (centralised) infrastructure, allowing the end users to manage node-hosted resources even when nodes are behind gateways or other network appliances implementing NAT, firewall rules, or any other restrictive policies. Among mechanisms enabling this flexibility, a key role is played by WebSocket-based tunneling, and either Google Cloud Messaging (GCM)- or WAMP-based messaging between the S4T lightning-rod and the *S4T IoTronic service*, the former being the point of contact with the Cloud infrastructure where the MCS application has to be deployed, the corresponding subsystem of which is the latter. *Web Application Messaging Protocol (WAMP)* [14], as a subprotocol, belongs to the IETF WebSocket (WS) standard, and defines some communication semantics for WS-transported messages, providing publish/subscribe (pub/sub) primitives, and remote procedure calls (RPC) as well, either simple or routed ones.

The S4T IoTronic service is designed as an OpenStack one, exposing the capability to manage one or more smart boards, remotely. This may be done either via a command-line interface, an *OpenStack CLI*, or a specific *S4T IoTronic CLI*, or even a Web browser though either set of REST APIs, as provided by the core OpenStack services and S4T IoTronic.

4. STACK4THINGS MCS PLATFORM

With respect to the node-side runtime, and specifically mobiles, in particular focusing on Android, there are the Sensing APIs and the environment-provided notification subsystem as key elements of the underlying platform at the basis of our design choices. The Sensing APIs are opaque in terms of our MCS platform, as any sensor tuning and reconfiguration requests are expected to be relayed to some environment-native subsystem managing this kind of requests, as the Sensing APIs in case of the Android ecosystem. The Android-native notification subsystem is useful to minimally involve provider-enabled Cloud-based mechanisms for push-based communication to devices, thus avoiding to track network conditions in general.

Google Cloud Messaging (GCM) [15], the most current Google-provided notification service, is leveraged for exchanging (bootstrapping) asynchronous messages in Android mobiles, an enabling step to support runtime customization mechanisms and other Cloud-initiated primitives, including on-demand activation of any WAMP-/WebSocket-based facilities, when not overly restricted in terms of limits on generated traffic. The choice of supporting the instantiation of WAMP-/WS-based channels, actually a custom communication bus, under a provider-supported mobile platform such as Android, stems from the inherent limitations (and costs) linked to the usage of (Android-native) GCM, which does not support significant payloads (e.g., file transfers), but is designed and marketed specifically for push notifications, and in our case employed also for bootstrap signaling.

Figure 3 shows the S4T MCS platform architecture in terms of the node-side components. The Sensing APIs provide the corresponding abstractions at the lowest level, upon which the node-side subsystem, called *S4T lightning-rod*, is built to arbitrate access to sensors/actuators, through a set of libraries for transducer virtualization, the *S4T virt libraries*, which provide interfaces for reading (and writing) sensors (and actuators, respectively), as well as settings parameters for devices' operating modes. Such operations are therefore available at the proper level of abstraction semantics. This means either locking or releasing resources in accordance to

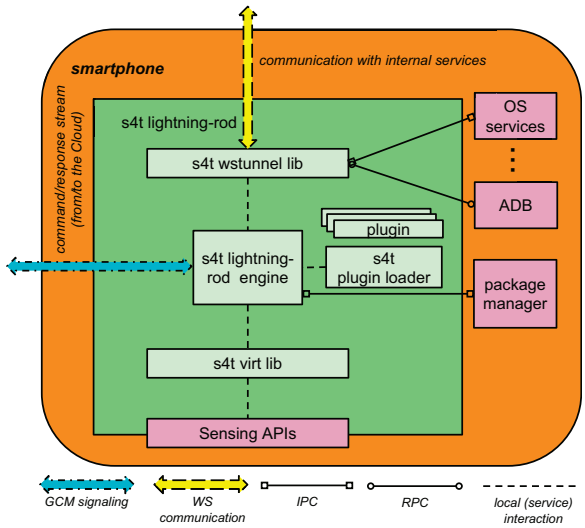


Figure 3. Stack4Things node-side MCS platform architecture.

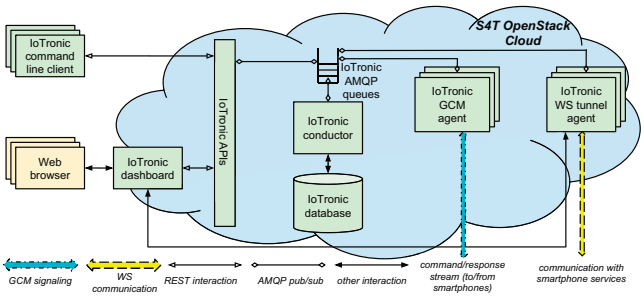


Figure 4. Stack4Things Cloud-side MCS platform architecture.

successful booking requests, and to constraints dependent on the APIs granularity, specific to the transducing resources.

The *S4T lightning-rod engine* sits at the core of the node-side software architecture, and communicates with the Cloud through a GCM-based messaging channel, at the very least, and, when needed, switching to a WebSocket-based full-duplex channel for WAMP messaging (see also Figure 4), in particular to support a wider range of functionalities, such as, e.g., sending data to (and receiving data from) the Cloud, or even executing commands input by the users via the Cloud, respectively. User-provided commands may be related to the interaction with the node-hosted resources (through the S4T virt libraries) and with operating system-level resources and tools (e.g., package manager, background services, filesystems). Bootstrapping communication with the Cloud is thus enabled by primitives for push-based messaging available through the platform-native SDK. Furthermore, WebSocket-based libraries (*S4T wstunnel libraries*) augment the engine to act as a WS-powered (reverse) tunneling server, connecting to a specific WS endpoint in the Cloud. This mechanism [16] enables internal services to be exposed through the tunnel for direct access by external users, whose incoming traffic is forwarded automatically to any internal service of choice running in the background. Outgoing traffic is captured and redirected into the tunnel and eventually reaches the end user that connects to the WS server in the Cloud, a user then ready, at last, to consume the node-hosted service.

The S4T lightning-rod subsystem also provides a *plugin loader*. By this component, custom plugins may be injected into the node environment from the Cloud, and loaded at runtime to expose specific (user-defined) primitives, including system-level interactions, albeit subject to sandboxing, such as, e.g., involving the system-native package manager, or any service automatically started at boot-time. The S4T Cloud-side MCS platform architecture (see Figure 4) features an OpenStack service newly designed for IoT, named *IoTronic*.

IoTronic is characterized by the standard architecture of an OpenStack service. The *IoTronic conductor* includes the core logic of the service, and manages the *IoTronic database*, which stores required information about IoT devices, e.g., unique identifiers of nodes, any ownership or delegation role nodes may obey to with respect to users and tenants, board properties and hardware/software characteristics. The conductor also dispatches RPCs among other components.

The *IoTronic APIs* exposes RESTful interfaces for end users. The OpenStack Horizon dashboard has been extended to expose all the functionalities provided by the IoTronic service and other S4T subsystems. The dashboard exposes also access to node-side services, relaying traffic from the user to the *IoTronic WS tunnel agent*. This agent acts as a controller for the WS endpoint to which nodes connect through wstunnel libraries.

Likewise, the *IoTronic GCM agent* works as a bridge for GCM-based communication between any node and other components. The command stream gets delivered by this agent during bootstrapping phases, as well as under situations of restrictions on traffic. It converts AMQP messages on the wire into GCM-compliant ones, and brokers from GCM to AMQP as well, conversely. In accordance to the current philosophy among OpenStack contributors, any communication occurring among Cloud-side subsystems, in this case IoTronic components, is implemented as pub/sub messaging over the network via AMQP queues. This enables the whole design to be as scalable as needed, as all components may be deployed on different hosts without affecting service behaviour. Also, multiple *tunnel agents* and *GCM/WAMP agents* may be instantiated, in that case each dealing with a subset of enrolled mobiles. Given all of the aforementioned options, high availability and redundancy may be guaranteed as well.

5. AN EXAMPLE

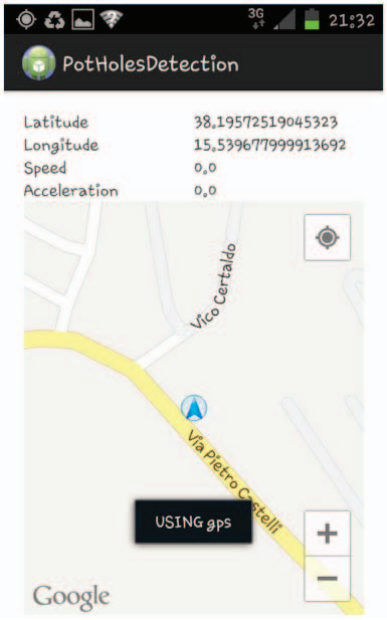
As an example of a S4T-powered MCS application, let us feature Smart City-like traffic service, specifically focusing on a Pothole Detection and Mapping (PDM). aiming at detecting and identifying road potholes through traveler mobiles. The PDM application is based on two components: an Android app, running on S4T-enabled mobiles and a central-

ized back-end collecting, filtering and preprocessing sensed data.

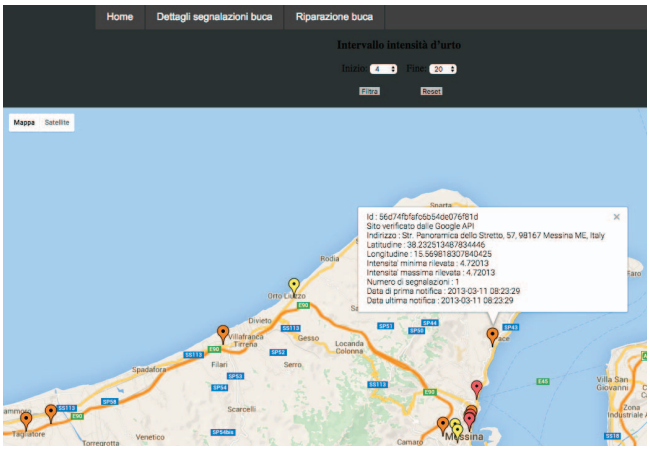
Aggregated data on geolocalized accelerometer samples from mobiles are used to create a first draft of the road map identifying potential potholes on the region of interest. This information is collected by the PDM app running in background on contributor devices to capture sudden changes on acceleration which could be related to a pothole. Such data are therefore locally evaluated and, in the case they match well-known (pothole) patterns, they are notified to the server side app together with their geospatial coordinates. Specifically, the measurements at the basis of the detection mechanisms are related to the accelerometer-provided acceleration vector, its (computed) euclidean norm and in particular the (impulsive) vertical (z-axis) displacement. Based on these input data, a Web-based application has been developed which, by interacting with the server-side database, implements preprocessing operations such as computing the differential values of the acceleration vector along the three axes of the frame of reference. This could be used to obtain further insights on the road segment condition highlighting issues in the roadway, which may be exploited by administrations to setup proper maintenance plan prioritizing on the extent of damage. To reduce the impact on mobile resources, data processing, mining and filtering are mainly moved to the Web server application, while just minimal pre-filtering operations, to prevent false positives, are implemented client side on mobiles.

A testbed composed of voluntarily contributing students and colleagues, going around Messina for almost a month, has been considered for the experiments. The results thus obtained are shown in Figure 5. In particular, Figure 5a depicts a screenshot of the app developed on Android devices, while Figure 5b shows the results collected by the above mentioned population in a month of experimentation, where the detected potholes are showed on a map.

The power of crowdsensing may be further appreciated by avoiding to centralize duties which may as well be performed client-side, actually just binding the processing task to the actor more intimately involved with the corresponding data, i.e., the contributor as data producer but also as volunteering compute node. Indeed this approach has been followed to actually fix imprecise locations of potholes. In particular, well-known limitations with precision in GPS positioning, typically on the order of a few meters at most, are not an issue when leveraged for navigation systems, due to the continuous tracking and the road maps as trails to follow. Yet, in the case of pothole detection, it is not trivial to identify with precision their positions, without resorting to huge amounts of (otherwise unneeded) data to be pushed and processed, to hook points of interest to specific segments, and ultimately infer the most plausible locations. If results are to be retained, albeit coarser-grained, a useful preliminary refinement may just consist in identifying the (shortest) segment among those originating from the (potential) pothole being detected, where those segments intersect neighboring roads perpendicularly. Fortunately there are already cloud-



(a) Android app



(b) Web app

Figure 5. PDM app screenshots

enabled service-oriented interfaces, like the Google Roads APIs, providing exactly this kind of information, i.e., nearest point, with respect to a location, belonging to a road. Considering the scale at which this kind of system could potentially operate, here crowdsensing comes to the rescue not only by leveraging mobiles as sources of interesting data, but also by offloading Cloud-side S4T subsystems, i.e., letting the mobile app itself query the APIs to infer road-compatible locations, before submitting two (related) datapoints, the raw (i.e., sampled) coordinates (for historical and analytics purposes only) together with the refined coordinates, the latter immediately ready to be leveraged for end-user visualisation on the dashboard.

In particular, this is done in our example by injecting and instantiating at runtime the corresponding code, tasked with querying the Roads APIs, as plugin through the aforementioned plugin loader. We can thus notice that our Cloud-enabled management of IoT infrastructure, coupled with a

plugin-based runtime customization system, enables us to leverage the crowd for sensing-related activities, augmenting results by offloading preliminary validation steps of the measurements to the crowd, just leaving long running tasks (e.g., analytics, storage) to the Cloud by default. This option turns out to be especially useful when time/space-local context, only available mobile-side, may be leveraged to preprocess results to be sent, as mentioned above. In particular, the preprocessing may involve other measurements to be sampled opportunistically, which would not overburden the Cloud anyway as this context would be discarded immediately after sending the (refined) sample to the Cloud. In this case, context may be other geocoordinates to be sampled for a brief interval after the main datapoint, in order to assess the confidence interval of the sampled coordinates, before querying the Roads API with a (possibly adjusted) position, to be hooked to the nearest road segment.

6. CONCLUSIONS

In this paper we explored MobileCrowdSensing (MCS) as a paradigm, especially in relation to IoT and edge computing, and adapted the Stack4Things (S4T) framework to account for basic mechanisms needed to enable MCS. This work in particular features a taxonomy and model of MCS, then presents S4T as an Infrastructure-as-a-Service framework for IoT and mobiles, amenable to be employed as a service-oriented platform for MCS applications, with specific focus on Android devices, in particular being able to instantiate and deploy at runtime custom code, useful to offload computation to the edge when deemed useful. These offloading capabilities have been put to use through an MCS application for mobiles, meant for crowdsourcing the mapping of road surface distress conditions, highlighting the potential of the envisioned approach.

REFERENCES

[1] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4Things: a Sensing-and-Actuation-as-a-Service framework for IoT and Cloud integration," *Annals of Telecommunications*, pp. 1–18, 2016.

[2] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, "A survey of mobile phone sensing," *IEEE Comm. Mag.*, vol. 48, no. 9, pp. 140–150, 2010.

[3] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *Workshop on World-Sensor-Web (WSW'06): Mobile Device Centric Sensor Net. and App.*, 2006, pp. 117–134.

[4] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *Pervasive Computing, IEEE*, vol. 7, no. 4, pp. 12–18, Oct 2008.

[5] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao, "Automatically characterizing places with opportunistic crowdsensing using smartphones," in *Proc. of the 2012 ACM Conf. on Ubiquitous Computing*, ser. UbiComp '12. New York, NY, USA: ACM, 2012, pp. 481–490.

[6] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell, "Bikenet: A mobile sensing system for cyclist experience mapping," *ACM Trans. Sen. Netw.*, vol. 6, no. 1, pp. 1–39, Jan. 2010.

[7] A. Hassani, P. D. Haghighi, and P. P. Jayaraman, "Context-aware recruitment scheme for opportunistic mobile crowdsensing," in *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*, Dec 2015, pp. 266–273.

[8] J. An, X. Gui, Z. Wang, J. Yang, and X. He, "A crowdsourcing assignment model based on mobile crowd sensing in the internet of things," *IEEE Internet of Things Journal*, vol. 2, no. 5, pp. 358–369, Oct 2015.

[9] B. Guo, D. Zhang, Z. Wang, Z. Yu, and X. Zhou, "Opportunistic iot: Exploring the harmonious interaction between human and the internet of things," *Journal of Network and Computer Applications*, vol. 36, no. 6, pp. 1531 – 1539, 2013.

[10] D. Zhao, H. Ma, S. Tang, and X.-Y. Li, "COUPON: A Cooperative Framework for Building Sensing Maps in Mobile Opportunistic Networks," *Parallel and Distributed Sys., IEEE Trans. on*, vol. 26, no. 2, pp. 392–402, Feb 2015.

[11] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4Things: an OpenStack-based framework for IoT," in *Future Internet of Things and Cloud (FiCloud), 2015 International Conference on*, Aug 2015, pp. –.

[12] S. Distefano, G. Merlino, and A. Puliafito, "A utility paradigm for IoT: the Sensing Cloud," *Pervasive and Mobile Computing*, no. 0, pp. –, 2014.

[13] ——, "Device-centric Sensing: an alternative to Data-centric approaches," *IEEE Systems Journal*, 2015.

[14] T. Oberstein and A. Goedde, "The Web Application Messaging Protocol," Working Draft, IETF Secretariat, Internet-Draft draft-oberstet-hybi-tavendo-wamp-02, October 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-oberstet-hybi-tavendo-wamp-02.txt

[15] "Google Cloud Messaging for Android." [Online]. Available: http://developer.android.com/google/gcm/index.html

[16] G. Merlino, D. Bruneo, F. Longo, S. Distefano, and A. Puliafito, "Cloud-based Network Virtualization: an IoT use case," in *Ad Hoc Networks*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, N. Mitton, M. E. Kantarci, A. Gallais, and S. Papavassiliou, Eds. Springer International Publishing, 2015, vol. 155, pp. 199–210.