Lessons learned from the design, implementation, and management of a smartphone-based crowdsourcing system

Adriano Faggiani
Dip. Ingegneria dell'Informazione
University of Pisa and IIT-CNR
Pisa, Italy
adriano.faggiani@iit.cnr.it

Enrico Gregori
IIT-CNR
Pisa, Italy
enrico.gregori@iit.cnr.it

Luciano Lenzini
Dip. Ingegneria dell'Informazione
University of Pisa
Pisa, Italy
l.lenzini@iet.unipi.it

Valerio Luconi
Dip. Ingegneria dell'Informazione
University of Pisa
Pisa, Italy
valerio.luconi@iet.unipi.it

Alessio Vecchio
Dip. Ingegneria dell'Informazione
University of Pisa
Pisa, Italy
a.vecchio@iet.unipi.it

ABSTRACT

Ubiquitousness of smartphones, when combined with the power of crowdsourcing, enables radically novel application scenarios, where a massive amount of mobile users scattered over wide geographical regions cooperate towards a single goal. Nevertheless these new possibilities come at the cost of additional complexity, such as the presence of humans in the control loop, scarce resources of mobile devices, increased management costs due the large number of users. In this paper we report and discuss the lessons learned from the design, implementation and management of Portolan, a smartphone-based crowdsourcing system aimed at monitoring large-scale networks.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring

Keywords

Crowdsourcing, network sensing, smartphone

1. INTRODUCTION

In the last decade the crowdsourcing paradigm emerged as a valid technique to solve large-scale problems: the key idea is to delegate to the "crowds" those tasks that are too costly or time consuming to be handled with traditional approaches. Thus, recently, crowdsourcing-based systems have been designed and implemented to face a variety of problems belonging to different domains. For instance, OpenStreetMap [1] builds a worldwide map through aggregation of crowd-generated content (GPS tracks and annotations); reCAPTCHA [12] exploits the contribution of humans to decipher distorted characters; SETI@home [2] uses the computing power of idle PCs to analyze radio signals coming from space. In the context of network monitoring, crowdsourcing demonstrated to be useful to detect network events, such as congestions or malfunctions, through correlation of measurements performed by multiple users [5].

Starting from these ideas we built Portolan, a smartphone-based crowdsourcing system aimed at sensing very large-scale networks, such as the Internet or cellular networks. Currently Portolan is able to: i) build maps of the Internet topology at the Autonomous System (AS) level of abstraction, and ii) to produce signal coverage maps of mobile operators. To accomplish such tasks, measurement activities are organized in a distributed fashion, to cover a large fraction of the target network in a reasonable time. In fact, crowdsourcing naturally supports fine-grained distribution of observation points, and it tolerates heavy monitoring loads through division and assignment of duties to a possibly large number of cooperating users. The choice of using smartphones as measuring agents has been driven by several factors. First, smartphones are able to visit different networks in a short time (both WiFi and cellular). As a consequence, a single device may provide different observations thanks to its mobility. Second, smartphones are always connected and they are on most of the time. This eases the automation of tasks and the assignment of measurement activities. Finally, smartphones may improve the analysis of networked systems from the geographical perspective, by associating a position, easily obtained through the embedded GPS unit, to the observed values.

In this paper we share our experience in building Portolan.

SENSEMINE'13, November 14 2013, Roma, Italy.

ACM 978-1-4503-2430-4/13/11...$15.00.
http://dx.doi.org/10.1145/2536714.2536717

In particular, we report some of the issues that have to be faced in designing and managing a smartphone-based crowdsourcing system and we discuss some possible solutions. Even though crowdsourcing provides important benefits when facing large-scale problems, the design and implementation of systems based on such paradigm poses additional challenges with respect to the implementation of traditional software systems. For example, system administrators do not have complete control over the devices that participate in the system, as smartphones are in control of their owners. The amount of data collected by the system is relevant and it is characterized by possible redundancy. Moreover, the adoption of smartphones introduces further difficulties, as they have limited resources in terms of computational power, battery, and bandwidth. Finally, a major element that contributes to the success of a crowdsourcing system is the number of cooperating users. Thus, motivating people to participate is crucial, and finding the way to stimulate users can be more challenging than building the system itself.

2. PORTOLAN: FUNCTIONALITIES AND ARCHITECTURE

To make the paper self-contained, this section provides a brief description of the Portolan's main functionalities and architecture.

As previously mentioned, Portolan is aimed at measuring and monitoring large networks. In particular, it can be used to *i*) provide a graph of the Internet at the AS level (using traceroute-like operations launched from users' smartphones), and to *ii*) collect georeferenced Received Signal Strength (RSS) samples, which can be used to produce signal coverage maps of cellular networks. Portolan has been designed according to a distributed client-server architecture. Users have to install an app on their smartphones to participate in the Portolan activities. The Android app is available for free at the Google Play store[1]. The Android app receives measurement tasks from the server and sends back the results once finished. The server aggregates the results to produce a complete view of the monitored networks. The overall architecture of Portolan is depicted in Figure 1. Since smartphones are characterized by scarce resources (battery, bandwidth, computing power), measurement campaigns submitted to the server are divided into smaller tasks. This way smartphones can execute the requested duty without compromising the users' experience. Assignment of monitoring activities to smartphones takes into account their geographic location, IP address, battery level, and network load. To obtain the requested scalability, the server has been itself organized as a distributed system where additional units, named *Proxies*, take care of the devices within a given region (measurements are location-driven and geographically limited). Communication between smartphones and server is based both on polling and push mechanisms. Push notifications have been implemented using *Google Cloud Message* (GCM), a notification service that makes possible to asynchronously contact Android smartphones. For further details about Portolan's architecture, the reader is forwarded to [7, 9].

[1]`https://play.google.com/store/apps/details?id=it.unipi.iet.portolan.traceroute`

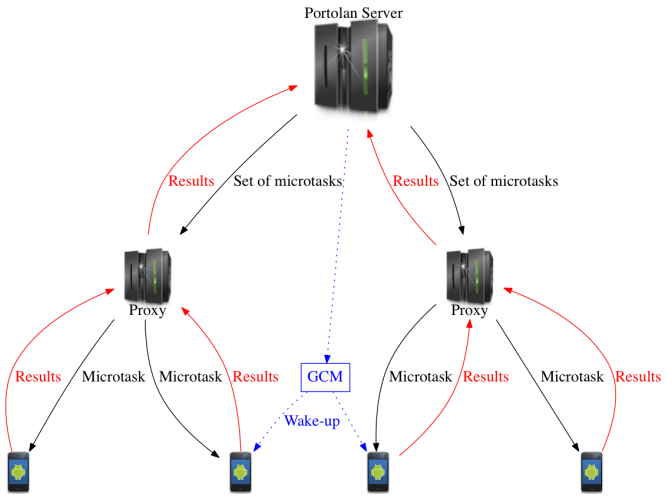


Figure 1: Portolan architecture

3. LESSONS LEARNED

In the following we summarize and discuss some of the problems we had to face.

3.1 Humans in the loop

Crowdsourcing applications are characterized by the presence of humans in the execution and control loop. This introduces a new set of problems with respect to more ordinary applications. In particular, the absence of a single controlling entity makes troublesome both management of software infrastructure and efficient production of data.

3.1.1 Management of infrastructure

During approximately six months of operation, we improved and extended the Portolan software running on the smartphone side. Changes concerned both bug fixes and the inclusion of new functionalities, as it is usually done for smartphone apps, where the presence of software distribution mechanisms (app stores) pushes towards a model characterized by a short update and release cycle. During the considered period, we released 8 versions of the Portolan app and every time we observed the same behavior: a large fraction of users update their app in a reasonable time (a week or so), whereas a small fraction is somehow "reluctant" to update the software to the new version. As a consequence, some terminals never got updated or got updated with a significant delay (even months) with respect to the release dates. Moreover, we had to face the co-existence of almost all the app versions released so far. Figure 2[2] shows the fraction of different versions of the Portolan app during the six months and Figure 3 highlights the current distribution of the app versions. The dates when the different versions have been published are reported in Table 1[3]. The presence of different versions of the same app is particularly troublesome if the app makes use of external services. For instance,

[2]Statistics extracted from the Google Play console. Google and the Google logo are registered trademarks of Google Inc., used with permission.

[3]Version 1.5 (number 6) and version numbers 9 and 10 have been used only for internal usage, and they never got published.

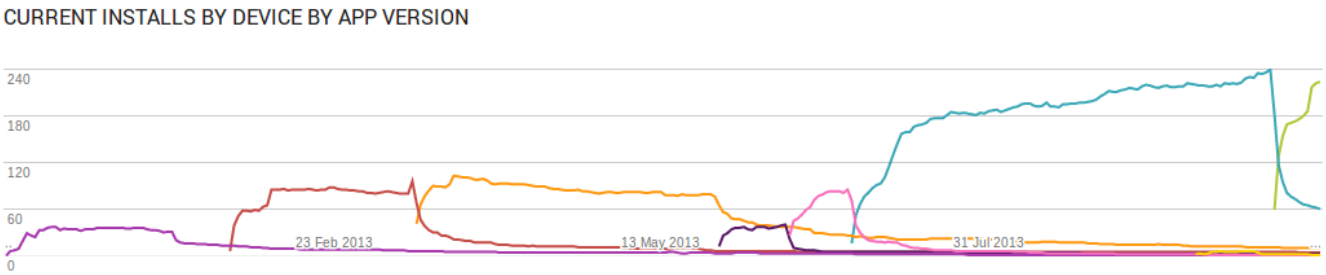


Figure 2: Different versions of the Portolan app[2]

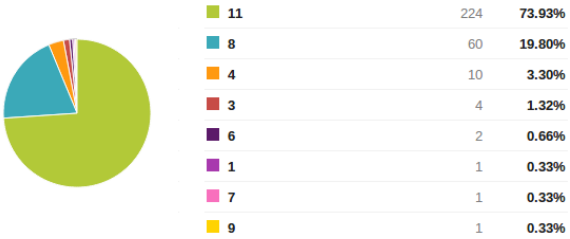


Figure 3: Current distribution of different versions of the Portolan app[2]

the update from 1.6 to 1.7 contained an improvement of the bandwidth estimation functionality, which relies on an external server. Since some users never upgraded their Portolan app to 1.7, we have been forced to keep active both versions of the external server to not compromise the functionality of the previous release. In the long term, these kinds of problems can greatly increase the complexity, and cost, of infrastructure management.

3.1.2 Production of data

In Portolan, wired networks are explored by using tools like ping and traceroute (or more efficient variations), whereas wireless networks are mapped and characterized as users wander in networked environments. The major problems that we registered are: 1) a task assigned to a given user may be dropped or aborted without notice; 2) users produce a significant amount of redundant information.

The first problem mostly depends on the volunteer contribution that is at the base of Portolan, and it may arise for a number of reasons. For instance, a user may uninstall the application just because he/she needs more space on his/her device or a task may be stopped because the device runs out of battery. The second problem, in Portolan, was caused by the repetitive behavior of human users: since wireless networks are mapped passively as the users move, some regions have been covered with excessive level of detail in a short period, while other regions have not been covered at all. In the end, this depends on the fact that the mapping task relies on the geographical position of the device, and this property is on the exclusive control of its owner. The same issue may arise whenever the task depends on properties that belong to the context of the user.

Version number	Version name	Release date
1	1.0	Dec 6, 2012
2	1.1	Jan 16, 2013
3	1.2	Jan 29, 2013
4	1.3	Mar 15, 2013
5	1.4	May 27, 2013
7	1.6	Jun 13, 2013
8	1.7	Jun 28, 2013
11	1.8	Oct 08, 2013

Table 1: Release dates and versions of the Portolan app

A possible solution to the first problem consists in assigning the same task multiple times to different users, with the hope that at least one of the involved users completes the requested duty. The drawback is, obviously, the waste of resources and production of redundant results. Facing the second problem is more complicated, as the context of users cannot be "forced" to the desired value. In general, having a very large user base makes this problem less stringent, as it increases the chance of having users who match the desired context properties.

In summary the components of a crowd are not completely interchangeable, and identifying the most suitable users to carry out a given task is not always easy, as this depends on a number of time- and position-dependent factors.

3.2 Constraints due to the smartphone platform

On the smartphone side, the design and implementation of Portolan proved to be rather challenging for two main reasons: *i*) mobile devices have limited resources, *ii*) operating systems for smartphones are constrained environment and some OS limitations make difficult the implementation of low level networking mechanisms.

3.2.1 Scarce resources

Despite continuous advances, hardware resources of smartphones are still limited. The most severe limitations are represented by bandwidth (communication can be expensive) and battery. Obviously, the programmer must be aware of the impact, in terms of energy and bandwidth consumption, of the solutions adopted during the design and implementation phases. In a crowdsourcing scenario, these constraints limit the workload that each device can carry out and push towards extreme parallelization of application ac-

tivities, where each device accomplishes a small fraction of the global task.

Monitoring of wired and wireless networks is, in general, not computationally intensive, as most of processing can be performed on the server side. Thus, the amount of energy spent to process data on mobile devices is usually negligible. Nevertheless, mapping of wired networks is performed actively and requires sending UDP probes and receiving the associated ICMP replies. This is expensive in terms of bandwidth and battery consumption (because the radio must be kept turned on). For this reason, in Portolan the wired mapping activity is limited to 200 traceroutes per day; this ensures that each smartphone consumes no more than 2 MB/day.

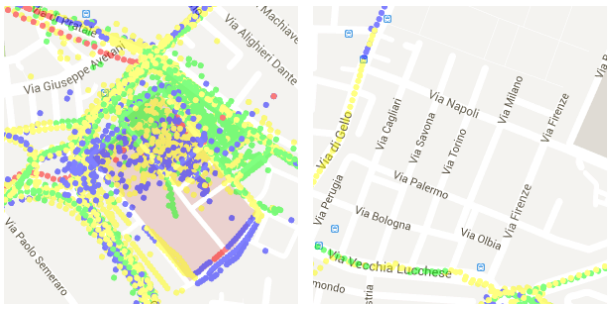
Collecting RSS samples is more demanding in terms of energy consumption. In fact, RSS samples must be georeferenced with great accuracy (to produce detailed coverage maps), and this forces to keep active the GPS module. In a first implementation, the RSS mapping activity was remotely started by the server (by assigning to the device a specific mapping task). To limit the energy consumption and not excessively deplete the user's battery this measurement had a duration of 30 minutes and each smartphone performed only 1 task per day. Despite this cautious set up, some users complained because they experienced an excessive battery consumption. Thus, in the subsequent versions of the Portolan app, we decided to not trigger RSS measurement remotely. Currently RSS mapping can be started in two different ways. A first possibility is using the passive Android location provider: if another app starts the GPS module, the geographic position is re-used by the Portolan app to georeference RSS samples. Alternatively, the user can manually start the RSS collection activity to trace the signal quality along a path he/she considers of interest; in this case the energy consumption is totally under control of the user.

Finally, if the battery level goes below 40% all Portolan activities are automatically suspended.

3.2.2 *OS limitations*

Operating systems for mobile devices are user oriented and they are optimized to ease the production of application-level software. In contrast, these OSes provide little support for the implementation of low-level custom mechanisms; this restriction became particularly evident during the implementation of traceroute-like tools as they require low-level networking primitives.

In Android, the main problem is represented by the available network API. The level of abstraction of offered functions is too high for the implementation of tools that require the production of manually crafted packets. To overcome this obstacle we had to make use of a native library implemented in C on top of the Linux kernel, which is at the base of Android. This allowed us to access sockets at the desired level, with all the necessary socket options that the implementation of tools like traceroute and ping requires. Another limitation, in Android, is represented by the unavailability of *raw sockets*. In Unix environments this type of sockets requires a superuser privilege level, but in Android all applications run at the user level. Thus an ICMP-based implementation of traceroute/ping is practically unfeasible. For this reason, we moved to an UDP-based implementation. Nevertheless, the inability to receive ICMP packets makes troublesome this solution as well. In the end, we have been able to implement a traceroute-like mechanism using the IP_RECVERR option of BSD sockets [7]. This option, when used with datagram sockets, appends all errors in a per-socket queue. In this way the programmer can retrieve all the information needed to compute the path traversed by UDP probes (in detail, when an ICMP packet is received, the programmer can obtain the type of ICMP packet and the source address of the host that generated such packet).



(a) Area densely sampled (b) Area scarcely sampled

Figure 4: Inhomogeneous distribution of RSS samples (Map data ©2013 Google)

From a more general perspective, the differences of the APIs provided by the available OSes (Android, iOS, Windows Phone) and the restrictions imposed to applications, in terms of privilege level and abstraction, suggest a careful evaluation, at an early stage, of possible implementation problems. This is particularly true when the mechanisms required by the crowdsourcing application operate at the lower layers of the software and networking stacks.

3.3 Division of workload

When studying a phenomenon, a certain degree of redundancy in the collected observations is desirable, for example to ensure reliability. In a crowdsourcing system redundancy is useful also to check if some users are producing erroneous information (maliciously or because of faults), in order to take adequate countermeasures. However, when adopting a crowdsourcing approach, the degree of redundancy can easily reach high levels, if it is not properly taken into account. For example, if multiple users observe the same phenomenon from the same observation point, the system could be forced to handle a large amount of redundant data.

In Portolan this problem became manifest when the collected RSS samples were analyzed. As mentioned, during an initial period, RSS measurement tasks were server triggered. Signal measurement campaigns were submitted to the Portolan server by human operators. The Portolan server then assigned signal measurement tasks to those smartphones that satisfied certain properties (the properties were static and specified at the time of campaign submission, such as the inclusion in a given region). During such period, the vast majority of Portolan participants were students and faculty

members of the University of Pisa, as the system was in a test phase. Thus, they were mainly grouped in the city of Pisa and in its surroundings.

After having collected RSS samples for approximately one month, results showed that the area of Pisa and its surroundings were extremely covered by signal measurements. What we did not expect is that several spots were showing a high density of samples and a significant overlap between samples, as highlighted by Figure 4a, whereas other areas were not covered at all, as shown in Figure 4b. This happened because the Portolan server, when assigning a RSS measurement task to a smartphone, did not consider the number of samples already collected in the surroundings of the smartphone position.

This example shows how desirable would be for a crowdsourcing system to have efficient strategies for dividing and assigning workload to participants, in order not to repeat the same observations multiple times without adding useful information.

Nevertheless, the implementation of such mechanisms in signal coverage mapping is not trivial for several reasons. First, smartphones communicate their position to the Portolan server only when they ask for available tasks, but a smartphone's position may change frequently and thus such information is characterized by rapid obsolescence. For example, at the moment of requesting a task, a smartphone could be in an area with a high density of samples, whereas few minutes later it could move to a poorly covered area (or vice-versa). The decision of assigning or not a given task should be based on more complex information such as the direction of the user or knowledge about his/her frequently visited places. Moreover, it is important to notice that the proximity of a smartphone to an area with a large number of samples does not imply that the exact position of the smartphone has already been sampled. For example a smartphone could be in a small street that has never been sampled, but contained in an area characterized by excessive coverage.

A possible solution could be to continuously monitor the smartphone's position and then start measurements when entering specific areas. However, continuous monitoring requires the GPS unit of the smartphone to be always on, and this would lead to significant battery consumption.

3.4 Prepare for big data

The main motivation that pushes to adopting the crowdsourcing paradigm is the need to carry out very large activities. This given, the success of a crowdsourcing system in accomplishing its goal obviously depends on having a very large user base. As far as Portolan is concerned, the main activities to be carried out are network measurement and monitoring. In particular, measurements must be taken from a large number of different observation points, and must be repeated over time, in order to analyze the evolution of the monitored phenomena. Hence, in the first few months of Portolan activity, hundreds of thousands of traceroutes were performed and stored in our database, and millions of RSS samples were collected, even with relatively few participating users.

To make the Portolan architecture as general as possible, and to ease the design and implementation of future measurement subsystems and extensions, we initially decided to make Portolan conform to the Open Geospatial Consortium (OGC) [10] Sensor Web Enablement (SWE) standard [4]. This specification defines XML interfaces for requesting measurements and retrieving results from single sensors and from sensor networks. Generally the SWE standard is used to access physical sensors [3, 8, 6, 11], however, due to its general specification, it can also be used for sensing properties that are not strictly physical, such as the bandwidth of a link or the topology of a network. Thus interaction between the Portolan server and smartphones has been organized according to the SWE specification. We also used an available implementation of SWE components to speed up the building process, in particular the Sensor Planning Service (SPS) for specifying and sending measurement campaigns to smartphones, and the Sensor Observation Service (SOS) for retrieving sampled data and storing samples from smartphones.

When the amount of data flowing into our databases reached a significant level, we noticed that some parts of the SWE specification were not as suitable for our purposes as we expected and that the implementation of the OGC standards was not completely customizable to fit our needs. In detail, we realized that the size of the data that we could extract from the database with a single request was too limited for our purposes. Moreover, the implementation did not allow us to specify all the filters that are instead included in the standard, thus limiting the amount and the quality of data that we could extract from the databases. This happened because the standard components that we decided to use were not explicitly designed for coping with such a large amount of data, being conceived for handling single sensors or small sensor networks, and because we underestimated the amount of data that could be produced by Portolan. As a consequence, we have been forced to replace the data extraction system with a customized and more efficient module. Practically, we substituted some OGC components with ad-hoc subsystems, in order to achieve the desired behavior.

The example highlights the importance of taking into account the size of the produced data from the early stages of the design phase of a crowdsourcing system. This is an issue of crucial relevance, if it is not properly handled it can lead to a long re-engineering phase of the system.

3.5 Motivating users

A crowdsourcing system with a small user base is of little use, thus motivating users to participate is fundamental for achieving the end goals. Nevertheless, reaching large numbers is not always easy as it involves aspects that, in some cases, go beyond the technical ones. Important elements include an effective publicity plan, and the inclusion of functionalities that are perceived as useful by the end users.

As far as Portolan is concerned, besides the initial involvement of students and faculty members of our institution, the publicity plan has been implemented through email-based advertisement. Messages describing the end goal of Portolan have been sent to some mailing lists of technical users in the area of networking. In particular, we stressed the sci-

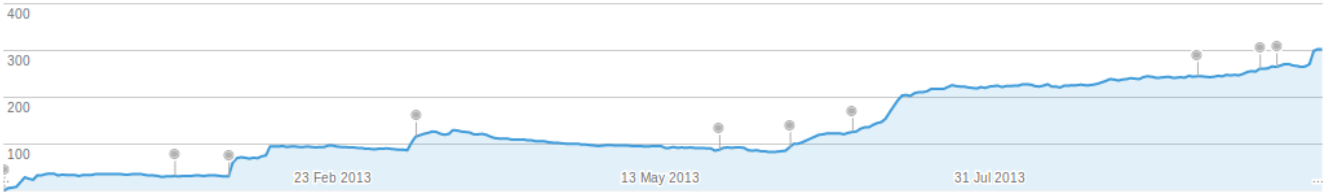


Figure 5: Number of active installs[2]

entific and non-profit purposes of Portolan, relying on the volunteer contribution of users belonging to the academic community.

Nevertheless, Portolan is somehow a niche product and this appeared particularly evident when we tried to involve non technical users. Conveying the ideas behind Portolan in a concise and simple form (as it is generally required for publicity) proved to be rather difficult. Thus, to stimulate the adoption from "normal" users, we included in the Portolan app some networking tools that can be directly started by the user independently from the Portolan activities: the traceroute tool and an estimator of the available bandwidth. The Android version of the Portolan client is currently available on Google play (for free), and the number of active users is now in the order of 300+. At the beginning the availability of the application was limited to Italy, as we preferred to evaluate the behavior and performance of the whole system under a light load. During such period the number of active installations did not increase very much, as it is shown in Figure 5. The installed user base started growing independently (without further publicity) when we widened the availability of the application to the rest of the world. Even if the steepness of the curve is not impressive, the general trend is positive and the user base is self-sustaining. The last three releases are the ones that included the new bandwidth estimation tool, and its impact appears to be positive.

To summarize, from this experience we learned that the adoption of the application from users may proceed at slower pace than expected and that motivating users is extremely important. Also, researchers must take into account that reaching the number of users needed for medium-scale experiments can be in the order of months, and that appropriate motivations, from the end users perspective, can reduce significantly this amount of time.

4. CONCLUSION

Designing, implementing and managing a smartphone-based crowdsourcing system is not trivial, as it involves technical problems originated by the limitations of the smartphone platform and by the large amount of users the system has to cope with. At the same time, the social aspects introduced by the crowdsourcing paradigm make the problem even harder. In this paper we shared our experience in building and managing a smartphone-based crowdsourcing system, in the hope that this discussion can be helpful to those researchers and practitioners who are going to design and implement similar systems.

5. REFERENCES

[1] OpenStreetMap. `http://www.openstreetmap.org/`.

[2] SETI@Home. `http://setiathome.berkeley.edu`.

[3] R. Aversa, M. Avvenuti, A. Cuomo, B. Di Martino, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, A. Vecchio, S. Venticinque, and U. Villano. The cloud@home project: Towards a new enhanced computing paradigm. *Lecture Notes in Computer Science*, 6586 LNCS:555–562, 2011.

[4] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens. New generation Sensor Web Enablement. *Sensors*, 11(3):2652–2699, 2011.

[5] David R. Choffnes, Fabián E. Bustamante, and Zihui Ge. Crowdsourcing service-level network event monitoring. *SIGCOMM Comput. Commun. Rev.*, 41(4):387–398, August 2011.

[6] A. Cuomo, G. Di Modica, S. Distefano, M. Rak, and A. Vecchio. The cloud@home architecture: Building a cloud infrastructure from volunteered resources. In *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011)*, pages 424–430, 2011.

[7] Adriano Faggiani, Enrico Gregori, L. Lenzini, Simone Mainardi, and Alessio Vecchio. On the feasibility of measuring the internet through smartphone-based crowdsourcing. In *Proceedings of the 10th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt 2012)*, pages 318–323, 2012.

[8] T. Foerster, B. Schaeffer, J. Brauner, and S. Jirka. Integrating ogc web processing services into geospatial mass-market applications. In *Proceedings of the International Conference on Advanced Geographic Information Systems and Web Services, GEOWS 2009*, pages 98–103, 2009.

[9] E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio. Sensing the internet through crowdsourcing. In *Proceedings of the IEEE Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 248–254, 2013.

[10] Open Geospatial Consortium Inc. `http://www.opengeospatial.org/`.

[11] C. Stasch, T. Foerster, C. Autermann, and E. Pebesma. Spatio-temporal aggregation of european air quality observations in the sensor web. *Computers and Geosciences*, 47:111–118, 2012.

[12] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, 321(5895):1465–1468, 2008.