Decentralised Approach for a Reusable Crowdsourcing Platform Utilising Standard Web Servers

Tenshi Hara*
tenshi.hara@tu-dresden.de

Thomas Springer*
thomas.springer@tu-dresden.de

Gerd Bombach
TUD Graduate Student
gerd.bombach@mailbox.tu-dresden.de

Alexander Schill*
alexander.schill@tu-dresden.de

*
Technische Universität Dresden
Faculty of Computer Science, Institute of Systems Architecture
Chair of Computer Networks, 01062 Dresden, Germany

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.

http://dx.doi.org/10.1145/2494091.2499574

Abstract
Crowdsourcing has gained increasing interest during the last years as means for solving complex tasks with the help of a flexible group of contributors. The crowd can contribute with collecting data in the field, completing map information or votes for ideas or products. Even though the participation of large numbers of users with heterogeneous devices in crowdsourcing is a highly recurrent task, generic infrastructures for crowdsourcing can be hardly found. Especially the management of users, mobile devices and contributed data has to be repetitively implemented in new projects. To ease the development of crowdsourcing applications, in this paper we propose a generic platform for crowdsourcing supporting diverse crowdsourcing scenarios, the ability to handle large numbers of users and the involvement of heterogeneous mobile devices. The evaluation is based on scalability and performance experiments in order to demonstrate the feasibility of our approach.

Author Keywords
Crowdsourcing, Location-based Services

ACM Classification Keywords
D.2.11 [Software Engineering]: Software Architectures—*Domain-specific architectures*

Introduction

Crowdsourcing presents itself to be a rather young concept in the field of computer science, emerging in the late 1990s and early 2000s, e.g. the *SETI@home* project was released to the public on 17 May 1999. Since then, manifold derivates of crowdsourcing have been investigated by many authors, e.g. in [3, 6]. In general, crowdsourcing can be considered a distributed process of providing ressources by a resource providing group of flexible size. This construct normally identifies two types of groups: The crowdfunders as the group interested in a ressource (e.g. processing time) and the crowdsourcers as the group providing the desired resource (e.g. idle processing time).

As diverse projects demonstrate, complex problems in many domains can be solved by exploiting the knowledge and abilities of users in the Internet. In all these systems users are involved explicitly or implicitly (e.g. by allowing the capturing of sensor data by their mobile devices in the background) in the crowdsourcing process.

All these systems require means for managing large user numbers, involving heterogeneous devices for data collection and the processing and verification of submitted data. Nevertheless, the potential for reusing crowdsourcing functionality and infrastructure components is rarely exploited. Most crowdsourcing projects focus on application level usage of user submissions, demonstrating the value of crowdsourcing in a multitude of domains. Generic infrastructures for crowdsourcing and documentation thereon can hardly be found.

When investigated in detail, e.g. in [1, 4], research focusses on the benefits of the flexibility of the group of crowdsourcers when using mobile devices such as smartphones, or the capabilities of involving human processing. In our opinion, one should not value human processing to highly, but consider crowdsourcing as an extension of the general problem of ressource availability and resource allocation which is present in any operating system. In this sense, crowdsourcing is reduced to a means of locating and allocating resources such as processing time, sensor access, actor control, et cetera, and becomes independent of the involvement of humans. Further, this process should be open to any kind of crowdsourcer as well as any kind of crowdfunder; hence, it should integrate any type of device capable of automated computation, not only servers *or* desktops *or* smartphones, but servers *and* desktops *and* smartphones.

The human factor can be reintroduced by extending the actual crowdsourcing process with social aspects such as a community with friends, awards, et cetera. Even the original SETI@home supported a competitive community by providing a credit system and the possibility to form competing teams, which return boosted the crowdsourcers' willingness to provide processing time.

In this paper, we present an approach for providing crowdsourcing as a service. The main contribution is twofold: First, we describe the concept of a generic, descentralised and highly scalable crowdsourcing platform which simplifies the development of crowdsourcing projects with special focus on supporting heterogeneous devices as well as any type of crowdsourcing with computable information. For its implementation standard web technologies were utilised. Second, we evaluate the crowdsourcing platform with respect to scalability and performance.

Challenges

Considered as the evaluation scenario of the proposed crowdsourcing architecture, location-based services can profit especially from crowdsourcing. *OpenStreetMap* is a

prominent example of a crowdsourcing approach utilising a steadily growing crowd of supporters, especially in well developed areas, map data of high quality could be aggregated within a very short time. According to [7], the aggregated data provides better information than comparable solutions of commercial tenderers. An extension to OpenStreetMap is the *indoorOSM* project[1], providing floor plans for indoor positioning and navigation. Floor plans can be created by users. Further examples of indoor projects are vast, e.g. OpenRoomMap [8].

Investigating existing solutions such as Amazon Mechanical Turk (AMT) or OpenTurk (OT), an extension of AMT providing additional functionality for user management paying respect to the users capabilities and an enhanced reward system, a clear demand for a reusable crowdsourcing platform can be identified. However, solutions – especially proprietary ones – generally tend to not provide information in detail on how users are managed and how crowdsourced data is stored, distributed and processed on the one hand; e.g. it is unclear how data and user information is aggregated, stored, distributed and processed within AMT. On the other hand, undisclosing solutions such as OT are specialised on specific crowdsourcing domains[2] and therefore do not address the general crowdsourcing demands which we want to address. Further, the problem of who owns aggregated data and the there from extracted information, as well as legal issues such as privacy and data protection are not sufficiently addressed, at least when applying German criteria.

Looking at the general problem of ascertaining required data and motivating the crowd to continue participating in the crowdsourcing process, the following challenges can be identified:

- client devices are used to ascertain the data,
- the crowd must be motivated,
- crowdfunding servers extract the required information from the crowdsourced data,
- users must be identifiable in order to provide rewards, ban malicious users, et cetera, and
- submitted data must be traceable, reproducible and verifiable.

Further, additional challenges can be identified by analysing usage characteristics of well-established crowdsourcing systems, such as Wikipedia, Kick-Starter, et cetera. Most in common is a demand for posterior modification or retraction of submissions on the users' side as well as a demand for deterring and posterior sifting of submissions on the crowdfunders' side.

Important Terms

We assume the reader is aware of the basic concepts of crowdsourcing. Additionally, we want to clarify some terms which are important with respect to our approach:

- *Crowdfunder*
  The person or entity requiring one or several resources that are not locally available, but distributed within a flexible group of system entities, the crowdsourcers.
- *User/Crowdsourcer*
  An individual member of the crowd using a service provided by the crowdfunder or an entity affiliated with the crowdfunder, producing crowdsourced resources as a side-effect of system usage or actively providing resources to the crowdsourcing process.

[1] http://indoorosm.uni-hd.de/
[2] OpenTurk specialises on academic crowdsourcing scenarios.

- *Client*
  A device used by the user in order to participate in the crowdsourcing process.
- *(Crowdsourced) Data*
  A subset of (crowdsourced) resources. – Any type of measurand ascertained and digitalised. Such measurands may be physical readings, numerical inputs, et cetera.
- *(Crowdsourced) Information*
  Any type of knowledge extractable from (crowdsourced) data. Either the information is directly contained in the data, or it can be computed as an indirect result from the data.
- *Submission*
  The dataset transmitted from the client containing data on or determining the crowdsourced resources, the user identification, a timestamp as well as some means for legal authenticity (e.g. an electronic signature).

In general, we assume clients to transmit crowdsourced data to the crowdfunders, whereas crowdfunders extract information from this data. The terms 'user', 'crowdsourcer' and 'client' will be used synonymously; however, one shall keep in mind that a crowdsourcer may use several clients, while a client in the majority of cases should only have one user.

Crowdsourcing Architecture

The identified challenges clearly show a demand for a reusable crowdsourcing service interconnecting crowdsourcing clients and crowdfunding servers. In the following section we give an overview of the platform and describe the details of major platform building blocks.

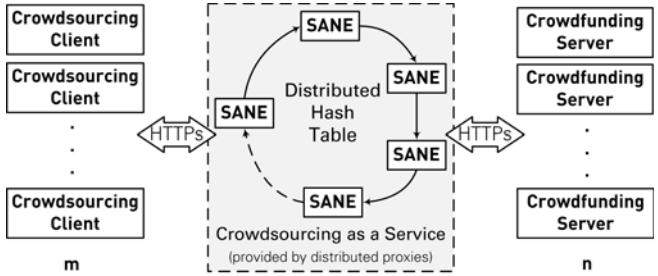

Figure 1: Architecture of the Crowdsourcing Platform

Platform Architecture
Following the goal to enable reuse of functionality for crowdsourcing, we propose a general crowdsourcing architecture providing the crowdsourcing platform as a service. An overview of the platform architecture is given in figure 1. The approach is based on a set of crowdsourcing proxies, called Server Access Network Entities (SANE) which act as intermediaries between Crowdsourcing Clients and Crowdfunding Servers. Thus, crowdsourcers are not directly connected to crowdfunders but submissions are always mediated by a SANE.

Designed as a generic component, the SANE is not be bound to a particular crowdsourcing derivate, but provide crowd and submission management for any derivate. From the crowd's perspective, members of the crowd are able to participate in several crowdsourcing processes at the same time using the same client device, while from the crowdfunders' perspective, submissions can be ascertained from a vast and variable crowd without having have to handle the crowd management. These aspects are indicated as m:n-mapping in figure 1.

In the following subsections we describe the details of our platform, namely the SANE architecture, the organisa-

tion of SANE components using a Distributed Hash Table and how to adopt the crowdsourcing platform in a crowdsourcing project.

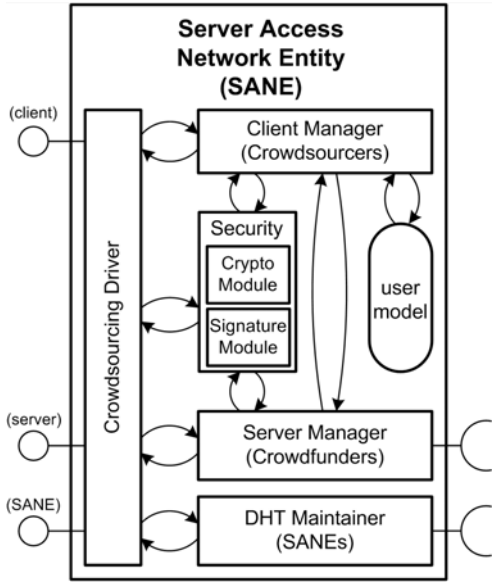

Figure 2: Architecture of the SANE

Server Access Network Entity
The task of a Server Access Network Entity is to handle the entire user, client and submission management for multiple crowdsourcing projects by ensuring anonymity, security, scalability, fault tolerance and performance. To cope with these requirements, the SANE consists of a set of modules as depicted in figure 2.

The *crowdsourcing driver* encapsulats any crowdsourced read and write-access oriented towards the serverside. It provides interfaces for crowdsourcing clients, crowdfunding servers and to interconnect SANE instances. In cooperation with a *security module* for encryption and signature-handling, the crowdsourcing driver forwards client related access to a *client manager*, and crowdfunding server related access to a *server manager*. The latter should conduct the actual call of setter methods on the crowdfunding server, while the client manager would conduct any access to stored client data, such as the user model representing the accessing client. As such accesses should be legally binding and reproducible as defined for the client access itself, it is necessary to ensure that all involved components make use of appropriate security modules, to the very least ensuring that all communication is signed and encrypted. In adition, the *DHT maintainer* module to enable a SANE proxy to participate in a group of SANE proxies forming a distributed crowdsourcing platform as described in the next section.

Maintaining anonymity of crowdsourcing submissions towards the crowdfunding servers is of imperative nature, as well. Therefore, any write-access to the crowdfunding servers is not only be proxied via the SANE, but also re-signed and re-encrypted, ensuring that any submitted data is traceable from the crowdfunding servers to the SANE, only. Nevertheless, the SANE should store any resigning and/or re-encryption, maintaining a legally binding reproducible trace of the data submitted.

Summarising, the SANE acts as a proxy for crowdsourcing read- and write-access between clients and crowdfunding servers, additionally storing all submissions of the clients redundantly. Obviously, the original crowdsouring data are stored on the crowdfunding servers. However, any submission is bound to a set of additional information, the identity of the client submitting, et cetera. These additional data must never be stored on the crowdfunding servers but is maintained in the *user model* of the SANE. Fur-

ther, the SANE shall be extensible for future community-related additions, such as a bridge to social communities, offering an extensible database for additional storage of pseudonyms, email addresses, avatar images, et cetera.

For obvious reasons, granted access rights on crowdfunding servers should be stored associated with user profiles. Therefore, the SANE not only stores client related information, but also lists of granted access rights in order to enable any client to gather information on the granted rights centrally.

Organisation of SANE Components
To ensure the scalability and fault tolerance of our crowdsourcing platform, the functionality is not provided by a single proxy, but by a set of distributed SANE components which cooperate to offer Crowdsourcing as a Service. This raises questions of how to organise proxies and how to assign crowdsourcing clients and crowdfunding servers to particular SANE instances.

For our crowdsourcing platform we follow the approach of proxies organising themselves utilising a distributed hash table (DHT). DHTs have been well researched and proven to be an efficient means of organisation of vast numbers of nodes. Hence, it is very simple for the crowdfunder to setup additional proxies, remove proxies or integrate a network with proxies of other crowdfunders.
For reasons of fault tolerance we suggest having SANEs neighbouring in the DHT act as backups for each other. The idea is depicted in figure 3, in which the ID-hash or 'client ID' respectively is the result of the hash computation on an exemplary 'myID' string. Within the DHT, SANE 3 handles the hash area corresponding to the client's ID-hash (solid clue); however, the neighbouring SANE 2 and SANE 4 act as immediate backups (dashed clues).

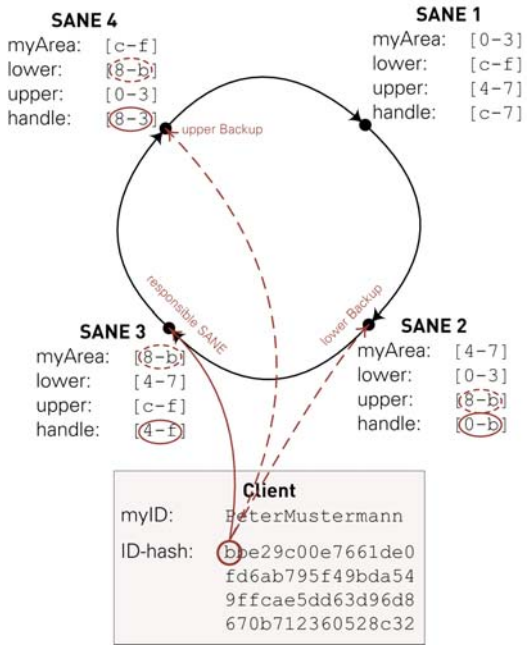

Figure 3: Exemplary DHT distribution of SANEs.

In our proof-of-concept implementation we utilised the IMEIs of the smartphone client. Which identification characteristic is used does not actually matter, it should only be unique. – Anyhow, we are aware of the security risks involved with basing IDs on an IMEI which is transmitted unencrypted in most mobile networks. However, for the proof-of-concept the IMEI was sufficient in order to automatically provide an unique identifier for each smartphone used in testing without having have to generate and assign usernames, et cetera. However, considering the described security and privacy issue as well as a possible demand to expand the user profiles of the crowdsourcers with additional aspects such as awards, avatar

images, et cetera, introducing an unique username and hashing over that username seems the simplest approach.

A general design problem arises in the simple DHT distribution not paying any respect to the users' geographic positions. However, it should be desirable to reduce communication distances, hence automatically reducing communication delays. When considering that Internet Protocol (IP) addresses are distributed following a well-defined geographic schema by the Internet Assigned Numbers Authority (IANA) and its regional internet registries, it would make sense to use any accessing client's IP address to determine their geographic location.

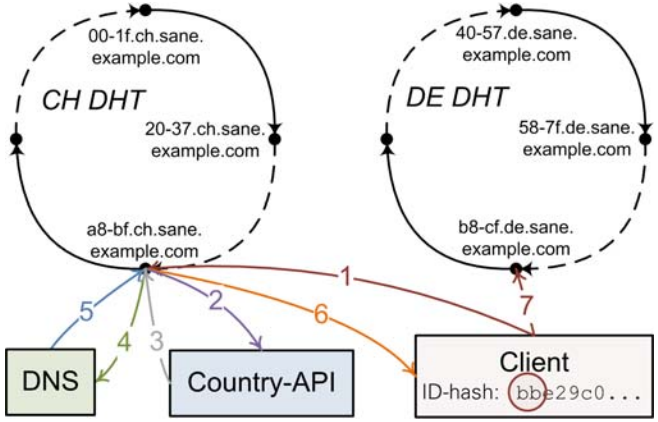

Figure 4: Only two client requests are required to lookup the geographically nearest SANE handling the client's hash area. The client is in Germany and not aware of local SANEs, but knows the SANE handling its hash area in Switzerland and contacts that SANE (1). The SANE determines that the client is in a different region and which region that is (2 and 3). Via the DNS the Swiss SANE locates the nearest German SANE (4 and 5) and informs the client (6). Finally, the client contacts the German SANE (7).

Therefore, it makes sense to divide the SANE distribution into regional hash areas, too. When further extending the now divided SANE distribution with a Domain Name System (DNS) interface, easy lookup of the next nearest SANE handling a client's hash becomes possible. For this, dynamic DNS host names can be utilised. The basic idea is illustrated in figure 4.

Adopting the Crowdsourcing Platform
Considering the beforehand introduced, developing a crowdsourcing project is very simple: The crowdsourcer defines which data is to be aggregated and implements an aggregation routine for the client device (e.g. logging of gyroscope data) as well as a processing routine on the server (e.g. extraction of movement patterns from gyroscope data). The rest is automatically handled by the SANE, namely the user and submission management, and user/client anonymisation.

To ease the setup process, we further suggest to utilise only freely available and standardised server technology, e.g. a server based on *L*inux, *A*pache, *M*ySQL and *P*HP (LAMP), and therefore also only standardised protocols.

Using communication based on the Hyper Text Transport Protocol (HTTP) and limiting it to POST and GET requests allows usage of existing infrastructures while at the same time no special preparations to firewalls and other network components are required when refraining from using 'exotic' ports. Taking this idea even a level lower, usage of XML or other special description languages can be limited to actual submission content, e.g. a set of sensor data. All other information, especially on success or failure of method calls, can be communicated by the means of HTTP itself, i.e. by RFC 2616 (Section 10) status codes. Using such low level means of communication, the focus should be shifted to POST requests,

For maximum compatibility with existing infrastructures and firewalls, standard HTTP communication as defined in RFC 2616 (Section 10) should be utilised.

Limitation to a subset of status codes – 200, 201, 202, 302, 303, 400, 403, 404, 409, 500, 501, 502 and 503 – accompanied by status code '424 `Failed Dependency`' from RFC 4918 (Section 11) is sufficient.

Status 424 is required in order not to simply deny access to SANEs for unregistered users and crowdfunders, but to inform of the necessity of prior registration.

as they transport variables within the HTTP header in contrast to GET requests where encryption of the request with the means of HTTPs is meaningless as the variables and their contents remain in plain-text within the Uniform Resource Link (URL) itself. This limitation to POST requests allows simple encryption by means of secure HTTP (HTTPs) without having the crowdfunder have to implement encryption themselves. Analysing the aspect of authenticated communication, the idea of placing signatures into the HTTP packets seems rather simple, but it is effective. – An example for the designed HTTP communication is presented in figure 5.

As mentioned, the entire aspect of the actual crowdsourcing management – e.g. provision of rewards to the crowd, holding back submissions for sifting, et cetera – is provided by the proxies, enabling crowdfunders with limited knowledge of or limited resources for crowd and submission management to ascertain crowdsourced data.

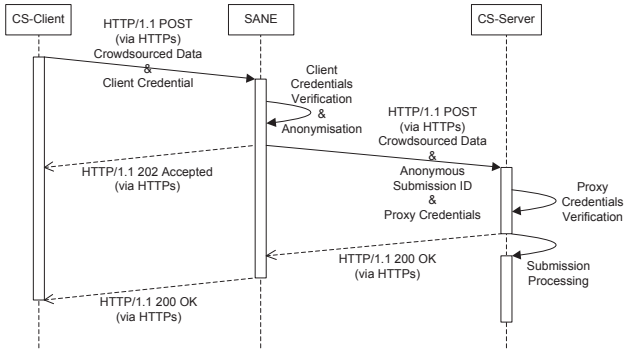

Figure 5: Exemplary communication sequence for a successful submission. Only standard HTTPs communication is utilised.

The proposed SANE unites the management of the crowd, the management of crowdsourced data and anonymisation features. Any individual from the crowd is identified on user and device level by unique credentials as well as a hash identifier equally distributing all users over the DHT. SANEs handling intersecting sets of the hash distribution exchange user/client data via asymetrically encrypted and signed HTTPs connections. Each device is assigned an unique submitter identification which is transmitted to the crowdfunder instead of the actual identification. Together with unique submission identifications, deterring and posterior sifting of submissions is enabled without disclosing the submitters identity to the crowdfunder. Inversly, via the unique submitter identification, rewarding and blocking of users is possible. Based on asymetric exncryption keys, a well defined group of maintainers is enabled to identify submitters in case of legal issues, e.g. police requests. Of course, any individual from the crowd can operate multiple devices; however, currently any device can only have one user[3].

Proof of Concept

The stipulated SANE was successfully implemented as a web server application in PHP (application logic) and MySQL (data storage), and deployed on a standard LAMP web server; in our test setting a cloud virtual private server (Cloud-VPS) with up to 3GHz CPU and 4GB RAM running Linux 2.6.32-41-server x64, Apache httpd 2.0, MySQL 5.1.63, and PHP 5.2.12-nmm4.

As memory consumption in means of RAM seems to be the bottleneck of most Apache servers rather than CPU speed, memory consumption was considered in the earliest stages of prototyping. The memory consumption of object-oriented PHP presented itself to be much higher

[3] Only after our proof-of-concept implementation was completed we were made aware of this detriment by the introdcution of Android 4.2 Jelly Bean in November 2012.

compared to structured PHP (a factor of 3 could be measured). This can be explained by the RESTfull request/reply philosophy of the communication at hand:

- Any communication between clients and SANEs is finalised with one request and one reply,
- in default, communication between SANEs and servers is finalised with one request and one reply; however, situation with two requests and replies are possible, and
- most of the communication between SANEs is finalised with one request and one reply.

Therefore, no demand to keep states, objects, et cetera over several requests and/or replies arises, especially as all objects created when handling a request are destroyed after finishing the computation of the reply. Even further, the computation strategy followed is linear; hence, object-orientation would have only created unnecessary overhead such as object control headers, object reference pointers, et cetera. In prevalent code classification, the strategy followed is classified as *imperative structured programming with modularised includes*[4]. Based in the proxy-nature of the architecture, the application logic required can be reduced to a database extender and the DHT organisation.

We implemented a crowdsourcing module for the MapBiquitous location-based service providing four use cases: ascertainment of WLAN and GSM fingerprints, and correction of positioning data based on those two. – As desired, the effort required from the crowdfunder in order to setup the crowdsourcing process could be reduced to

- the provision of the actual crowdfunding server processing the crowdsourced data,
- the provision of the ascertainment logic on the crowdsourcing clients, and
- the definition of data to be ascertained and the necessary data types thereof.

In our proof-of-concept the crowdfunding server (a WFS server) already existed; hence, only the corwdsourcing application logic for extraction of information from the crowdsourced data, as well as communication with the SANEs required implementation. The latter was a straight forward realisation of the defined interface in HTTP.

A mobile device MapBiquitous application for the Android operating system existed anyway, so merely the ascertainment had to be added. Communication to the SANEs could be 'implemented' by adding a SANE communication library to the Android application.

On the three SANEs deployed, the same crowdsourcing module could be uploaded without any modification. This module consisted of an interface definition containing the required variables (i.e. names and types of data to be ascertained) and a definition of immediate variable tests to be conducted on the SANEs, e.g. check whether a provided WGS 84 GPS position[5] was valid. The entire rest (i.e. user and client management, et cetera) was usable out-of-the-box from the deployed SANEs.

Within our proof-of-concept implementation the interface definition itself was a serialised PHP array of the data definitions. This serialisation had to be provided to the SANE module as is; however, for an actual deployment a user-friendly frontend for the definition process is recommendable.

[4] The '*modularised includes*'-part originates in the fact that the code is organised into modules which are included on demand.

[5] Global Positioning Ssystem coordinates are most commonly procided in World Geodetic System (1984) format.

In summary, our proof-of-concept implementation was able to easily prove the ease of crowdsourcing setup effort for the crowdfunder.

Evaluation

We followed several concepts of solid evaluation to determine feasibility of our proposal. From these, the feasibility of crowdsourcing was investigated in context of optimised positioning of the location-based service MapBiquitous in [2]. The performance and scalability of the proposed SANE was investigated in [5]. Within this paper, we want to focus on the 'standard web servers' aspect of our approach by providing the evaluation results for the Apache setup used.

The performance and scalability of our implementation introduced earlier proved to follow the same characteristics as any website deployed on comparable servers. However, as it is hardly possible to measure scalability with hard numbers since physical measurands are strongly correlated to the hardware used, the connection speed, et cetera, a comparative approach of evaluation seemed to be the method of choice. Therefore, we compared our implementation with websites operating on the same or very similar hardware. In terms of propagation the scalability could then be estimated. – Measuring series for our test setting were conducted for different amounts of parallel client requests and were repeated ten times each. The results were then averaged into one representative table. The results for the test setting are visualised in figure 6 and clearly show a strictly linear correlation between the amount of parallel client requests and the packet losses (short-dashed, blue line) as well as the successful replies (solid, green '200 OK' line as well as long-dashed, red '502 Bad Gateway' line). The saturation at about 5000 parallel competing requests breaks the linear correlation and is due to the underlying Apache httpd web server failing at that high degree of parallelism. – Note, the packets labelled '502 Bad Gateway' are returned to clients whenever the request from the contacted SANE to crowdfunding server is unsuccessful; however, the original request from the client to the SANE must be considered successful as it is processed and a valid HTTP status code is returned.

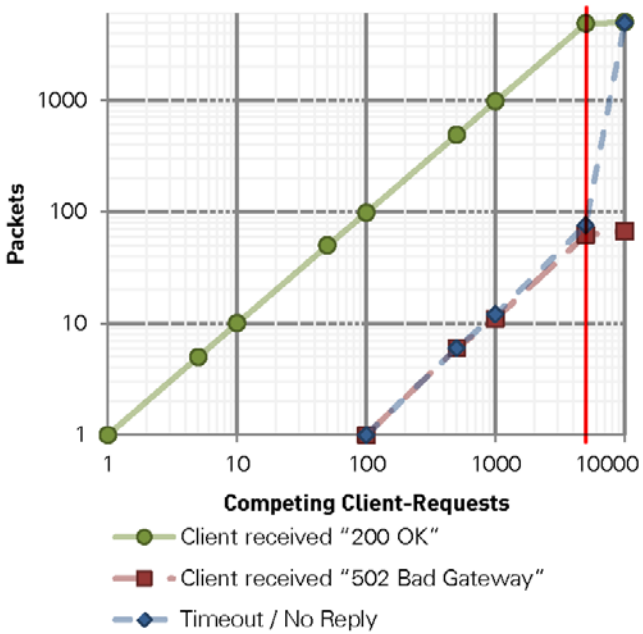

Figure 6: Packet transmission statistics for the performance and scalability test.

These results match expectations towards the underlying web server and prove that the conceived SANE can be considered a website deliverable by standard web servers.

Even further, the results can be matched to any website with database usage and HTTPs communication on underlying web servers with similar hardware specifications as the database tasks at hand are comparable to those of a content management system (CMS) or forum: Check whether a user exists, check their access rights, retrieve or store a submission, et cetera.

In fact, our database structure is very comparable to that of a CMS with a forum – such as Joomla or Wordpress – with the only exception that it does not store textual posts and threads consisting of posts, but crowdsourced submissions (which compare to the posts) as well as crowdsourcing processes (which compare to the threads) encapsulating those submissions. – At no time during our tests did the database turn into a bottleneck.

Additionally, however not unexpected, the results prove that the architectural design of a network entity similar to a website – i.e. using HTTPs communication with regular HTTP packets – is totally compatible with existing infrastructures. At no time demand for modification of firewalls, et cetera had arisen. Even further, correct functionality of interfaces could be controlled with standard web browsers such as Firefox, so no special tools were required.

Originating in the proxy-like design of the architecture, no significant[6] additional communication data overhead emerged for clients utilising the SANE architecture for anonymisation purposes. However very expectedly, a proxy delay time required to actually proxy the communication is inevitable as the anonymisation of the clients require that the entire connection is decrypted and verified as well as signed encrypted anew within the SANE. Due to the interface definitions present on the SANEs, any request normally addressed towards a (crowdfunding) server could be proxied via a SANE and therefore concealing the originating client's identity from the server.

[6] Only the identification token, a timestamp and the appropriate signature had to be added to the request.

Outlook

By the means of crowdsourcing complex tasks in multiple application domains can be solved. However, even if there is a large set of commonalities shared in many projects, generic infrastructures introducing reusable crowdsourcing functionality as well as user and submission management can hardly be found. Additionally crowdsourcing is mainly investigated at application level.

The proposed generic crowdsourcing platform relieves crowdfunders from recurrent tasks of user and device management, submission management, anonymisation, traceability of crowdsourced data, et cetera with utilisation of simple existing technologies available on standard web servers. Thus, development of crowdsourcing projects can be simplified and reduced to the definition of data to be ascertained and implementation of ascertainment applications for the involved client devices.

Currently, the SANE architecture is used by the Map-Biquitous location-based service to crowdsource WLAN and GSM fingerprints, correct data of indoor positions, and provide anonymisation to users not participating in the actual crowdsourcing process.

In further research work and student theses we extend the SANE architecture's utilisation to an automated road condition monitor for cyclists based on gyroscope and positioning data, as well as an automated canteen queue monitor based on WLAN client count and sound level.

Acknowledgments

The authors wish to thank the many contributors of the MapBiquitous project. Jan Scholze developed the concepts and prototype of the initial system and desktop client during his student and master thesis. Students of the practical course on development of mobile and distributed systems, Kay Werner, Christoph Keßler and Sina Grunau implemented and evaluated the initial Android prototype and server-side components, enabling easy extension towards crowdsourcing.

References

[1] Alonso, O. 'Perspectives on Infrastructure for Crowdsourcing'. *Workshop on Crowdsourcing for Search and Data Mining* (February 2011). CSDM 2011 @ WSDM 2011.

[2] Bombach, G. 'Untersuchung von Methoden des expliziten CrowdSourcing für Location-based Services in MapBiquitous'. Assignment Paper, TU Dresden, December 2012.

[3] Brabham, D. 'Crowdsourcing as a Model for Problem Solving – An Introduction and Cases'. *Convergence: The International Journal of Research into New Media Technologies 14*, 1 (2008).

[4] Chatzimilioudis, G., Konstantinidis, A., Laoudias, C., and Zeinalipour-Yazti, D. 'Crowdsourcing with Smartphones'. *IEEE Internet Computing (IC '12) 16*, Special Issue (September/October 2012), 36–44.

[5] Hara, T. 'Towards a reliable Architecture for Crowdsourcing in the Context of the MapBiquitous Project'. Thesis, TU Dresden, October 2012.

[6] Howe, J. *'Crowdsourcing: Why the Power of the Crowd is driving the Future of Business'*, first paperback edition ed. Crown Business, September 2009.

[7] Neis, P., Zielstra, D., and Zipf, A. 'The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007-2011'. *Future Internet 4*, 1 (12 2011), 1–21.

[8] Rice, A. C., and Woodman, O. J. 'Crowdsourcing world models with OpenRoomMap'. In *PerCom Workshops* (2010), 764–767.