Lowering the Barriers to Large-Scale Mobile Crowdsensing

Yu Xiao
Carnegie Mellon University
Aalto University

Pieter Simoens
Carnegie Mellon University
Ghent University - IBBT

Padmanabhan Pillai
Intel Labs

Kiryong Ha
Carnegie Mellon University

Mahadev Satyanarayanan
Carnegie Mellon University

ABSTRACT

Mobile crowdsensing is becoming a vital technique for environment monitoring, infrastructure management, and social computing. However, deploying mobile crowdsensing applications in large-scale environments is not a trivial task. It creates a tremendous burden on application developers as well as mobile users. In this paper we try to reveal the barriers hampering the scale-up of mobile crowdsensing applications, and to offer our initial thoughts on the potential solutions to lowering the barriers.

1. INTRODUCTION

In recent years, there has been phenomenal growth in the richness and diversity of sensors on smartphones. It is now common to find two cameras, a GPS module, an accelerometer, a digital compass, a gyroscope and a light sensor in a single smartphone. And there is more to come! The rich information about the smartphone user's activity and environment provided by these sensors inspired the first wave of sensing applications that personalized user experience based on the sensed context. Now, a second wave of mobile sensing applications is gaining momentum. The focus has shifted from individual sensing towards *crowdsensing,* defined as "individuals with sensing and computing devices collectively sharing information to measure and map phenoma of common interest" [8]. Initially, crowdsensed inputs were analyzed offline, for example in the analysis of transportation activities in urban spaces [31], for the measurement of interperson similarity [12], or for mental and physical health assessment of elder people [20]. In more recent crowdsensing applications, the collected inputs are processed in real time. Examples include traffic monitoring [32, 33], public safety management [24], and collaborative searching [28].

A hypothetical use case serves to illustrate the potential benefits of crowdsensing using information-rich multimedia sensors and some potential pitfalls [22]. Imagine that a small child gets lost while watching a parade in the middle of a large city. The distraught parents, upon noticing their child is missing, immediately use their smartphone to initiate a search, providing sample images with their child's face. A crowdsensing search application tries to match these with the videos and images being captured by the many smartphone cameras in the crowd. Any potential matches are forwarded to the parents' phone, along with GPS location information. With thousands of electronic eyes applied to this problem, the child is quickly found, before she herself is even aware of being lost. For this use case, the large number of smartphone cameras in use makes it likely the child appears in one or more captured images; however, the crowdsensing search application itself can succeed only if a sufficiently large number of smartphone users participate.

More generally, there is a growing realization that *scale* is the key to the success of crowdsensing applications. Since individual users may go offline and individual sensor readings may be inaccurate or corrupted, the reliability and trustworthiness of crowdsensing applications scales more than proportionally with the number of users. Access to a vast user base is thus crucial. However, our survey of the literature shows that today's mobile crowdsensing applications using physical sensors like GPS have rarely been scale up to more than 1,000 participants.

Table 1 shows a representative sample of crowdsensing studies. Much to our surprise, the number of participants is often omitted in the papers reporting these studies. When concrete numbers are provided, the crowd sizes are usually small. It is only with data sources that are easy to collect (e.g. from social networking applications such as Twitter) that larger crowds have been studied. The one notable exception is the work of Balan et al [2], discussed in Section 2.

What limits the scaling of crowdsensing applications? In this paper, we explore this issue and and propose an architectural solution. We then explore the merits of this architecture, and discuss potential implementation challenges.

2. OBSTACLES TO CROWD SCALING

Crowdsensing applications, including the ones that exist today and the emerging class of applications making use of richer multimedia sensors, face three major barriers to achieving the large crowd sizes critical to their success.

The first obstacle is the heterogeneity of sensing hardware and mobile platforms. In today's mobile device market, there are at least three popular software platforms, includ-

Reference	Mobile Platform	Application Category	Crowd Size	Input
Zhou et al. [32] (Mobisys 2012)	Android	Transportation	unknown	cell tower ID, audio signal accelerometer
Tiramisu [33] (CHI 2011)	iOS	Transportation	28	GPS
SignalGuru [10] (MobiSys 2011)	iOS	Transportation	13	video frames
Balan et al. [2](Mobisys 2011)	Car GPS	Transportation	15000	GPS
Mathur et al. [14] (Mobisys 2010)	Car GPS	Transportation	500	GPS
Niu et al. [17] (Com.geo 2011)	Blackberry	Transportation	unknown	GPS
Bao et al. [3] (Mobisys 2010)	Symbian, iPod	Social Application	unknown	video
Wirz et al. [27] (SCI 2011)	Android	Social Application	unknown	GPS
CrowdSearch [28] (Mobisys 2010)	iOS	Search	unknown	image
GeoLife [31] (WWW 2009)	GPS phones	User Behavior Study	107	GPS
SoundSense [13] (Mobisys 2009)	iOS	User Behavior Study	unknown	audio stream
#EpicPlay [25] (CHI 2012)	Twitter	Social Application	unknown	tweets
Wakamiya et al. [26] (ICUIMC 2012)	Twitter	User Behavior Study	39898	tweets
Fujisaka et al. [7](ICUIMC 2012)	Twitter	User Behavior Study	8139	tweets
CrowdSearcher [4] (WWW 2012)	Facebook	Search	137	text

Table 1: Representative Sample of Crowd-sensing Applications

ing Android, iOS and Windows 8. Applications written for any of these can not be run on the others. Even different versions of a particular platform are sometimes incompatible, due to changes in hardware or evolution of the software APIs. Furthermore, the Apps model in vogue today, along with the relatively low processing power of mobile devices, has encouraged smaller, stand-alone applications, and discouraged the development of external libraries, middleware, and virtualization techniques to bridge the differences between platforms. There is no sign that a single platform will dominate this fragmented market in near future. For true ubiquity, application developers need to write, test, support, and maintain versions of their applications for all of these platforms. In a sense, the complexity of the crowdsensing application space grows with cross product of the number of platforms and the number of applications.

This issue of heterogeneity is underscored by the experience of Balan et al. [2], who conducted one of the largest crowdsensing studies to date. It took them six months to deploy one version of their GPS-based crowdsensing application on 15,000 taxis in Singapore, mainly due to the heterogeneity of the on-car GPS devices provided by different vendors. Web-based applications implemented in HTML5 are sometimes put forward as a "write once, run everywhere" alternative. Unfortunately, support is generally limited to the lowest common denominator, and direct access of the underlying sensing hardware is disallowed.

The second obstacle is the burden today's crowdsensing applications place on users. Today, each user must install a separate proprietary application for every crowdsensed experiment in which s/he wishes to participate. As a result, the deployment of a single crowdsensing application is limited by the rate at which users adopt and install it on their devices. It can take weeks or months for a newly introduced application to reach the critcial mass of participants needed for it to be useful. Rapid, large-scale deployment, as in the lost child usage scenario above, is impossible with an install-based deployment model. Users also have to be tolerant of the processing, memory, and battery life these applications consume. Because today's mobile operating systems are designed to shield applications from each other, each application is meant to be self-contained and does not share information with others. In addition, some sensors, such as cameras, need to be exclusively locked before use. Participating in more than one crowdsensing application at a time is therefore not easy, even if a user is positively inclined.

The third obstacle, which primarily affects future applications, is the increasing network bandwidth demands of emerging crowdsensing applications. Table 1 shows that the GPS data has been the most widely used sensing information in the existing crowdsensing applications. However, looking ahead, we envision growing use of data-rich, multimedia sensing information like video [1] in emerging applications such as augmented reality, or the video-based lost child locator discussed above. These applications not only demand far more computing power, but also far more network bandwidths to send data to the cloud infrastructure. Based on data rate analysis of 80 videos on YouTube captured from a first-person viewpoint, each participant in a video-based crowdsensing application will upload between 0.6 Mbps (360p resolution) and 5.6 Mbps (1080p resolution). With many users, such an application can easily overwhelm link capacity in regional networks and into datacenters. For example, Verizon recently introduced state-of-the-art 100 Gbps links in their metro networks [18], yet these are only capable of supporting 1080p streams from just 18000 users. A broadly-deployed application with 1 million users will require 1–2 Tbps, 200x the total upload bandwidth of all YouTube contributors [30] today. An application model where each device sends data to centralized servers (as is typical today) cannot scale to support data-rich sensors. Ensuring the scalability of crowdsensing with data-rich sensors requires rethinking application and cloud architectures to acquire, process, and aggregate such data efficiently.

Ultimately, all three of these obstacles are ramifications of the deployment model in vogue today, where participation in each crowdsensing activity requires a separate application that must be installed and run on user devices, and directly communicates to central servers. To overcome these obstacles, we must rethink the structure and deployment model used in crowdsensing applications.

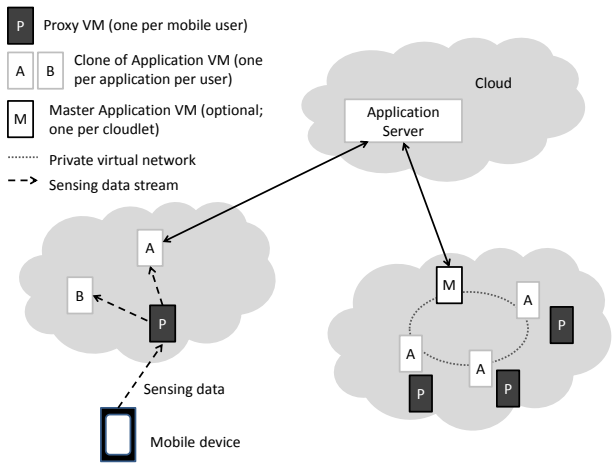

Figure 1: System Architecture.

3. PROPOSED SOLUTION

We propose a crowdsensing deployment model built around 3 core design principles:

- separation of data collection and sharing from application-specific logic.
- removal of app installation on smartphones from the critical path of application deployment.
- decentralization of processing, and data aggregation near the source of data.

Each of these design principles addresses one of the obstacles discussed above. By construction, our proposed solution overcomes the key barriers to scaling up crowdsensing applications.

3.1 System Architecture

The 3-tier system architecture of our deployment model is illustrated in Fig. 1. The first layer is composed of the mobile devices, whose roles are essentially reduced to that of (multi-input) sensors forwarding captured data to *proxy VMs* in the second layer. The second layer comprises of distributed cloud infrastructure deployed close to the users, typically in the access or aggregation network of network providers. The concept of distributed cloud infrastructure here is akin to the concept of *cloudlet* presented in [23]. In practice, this can be a private cloud owned by a business or community, or a small data center such as Myoonet's Micro data center [15] that is deployed by a cloud operator. For the sake of simplicity, we will refer to this distributed cloud infrastructure as cloudlets in the remainder of this paper.

Each proxy VM is associated with a single smartphone, and is kept physically close to the user through VM migration to other cloudlets or public clouds. This ensures network resources to transfer data from the mobile device is minimized. The proxy VM handles all the requests for sensor data on behalf of the mobile device. It is essentially an extention of the mobile device into the cloud, and can perform custom data preprocessing, e.g., to enforce privacy settings or handle quirks of the mobile platform, and enforce user preferences on data sharing and crowd participation. From here, data is forwarded to one or more *application VMs* also running on the cloudlet infrastructure.

Application VMs perform data processing steps specific to each crowdsensing application. Each application VM hosts a single crowdsensing application, which is not customized to any particular mobile platform. Generally, for each crowdsensing activity, one application VM is assigned to each participant, making it easy to migrate a user's proxy VM together with the associated application VMs, preserving any hard state they may contain. If an application does not need to maintain hard state for each user, then a single application VM can be shared by all users on a particular cloudlet.

The application VMs for each sensing service are deployed by a coordinating entity on the highest layer in our architecture, typically by the application server running on the centralized cloud infrastructure. In practice, when many application VMs are run on each cloudlet, the application server can initiate a master application VM (MAVM) on each cloudlet and delegate managment and data aggregation tasks. The MAVM will coordinate, clone, and configure the application VMs on the cloudlet, and aggregate data within the cloudlet before forwarding results. Depending on the application, the MAVMs on multiple cloudlets may form a peer-to-peer overlay network / tree to scalably aggregate data to the central application server.

Our deployment model is predicated on two assumptions. First, this architecture depends on distributed cloud infrastructure near the user. We are not the first to propose this, and a consensus has grown in the research community that offloading to nearby computing infrastructure (cyber foraging) is needed for compute-intensive mobile applications. The vision of executing customized VMs on nearby infrastructure has been articulated many times, e.g. in [6] and [9]. Our concept of distributed cloud infrastructure to host proxy and application VMs fits perfectly in this vision.

Second, our approach assumes a standard API exists for the data transfer between the proxy VM and the associated application VMs. However, we argue this is a much easier task to accomplish than having to write an individual application for each mobile platform (and possibly for each individual version of the mobile platform). Indeed, the output of scalar sensors can be represented as a few integers (e.g. GPS coordinates, temperature value, ...), and standards for multimedia data (e.g., video formats) already exist. Combining such data with standardized XML format descriptions, one can establish a standard for communication between proxy VMs and application VMs. In fact, several programming frameworks for crowdsensing applications have proposed solutions to abstract sensing information [29, 21] and task description [19]. These programming frameworks can be leveraged in our model as well.

3.2 Crowd Bootstrap

Let's revisit the lost child scenario from Section 1 to see how a crowdsensing task can be rapidly bootstrapped using our deployment model. As shown in Fig. 2, the process of crowd bootstrap can be summarized in the following seven steps.

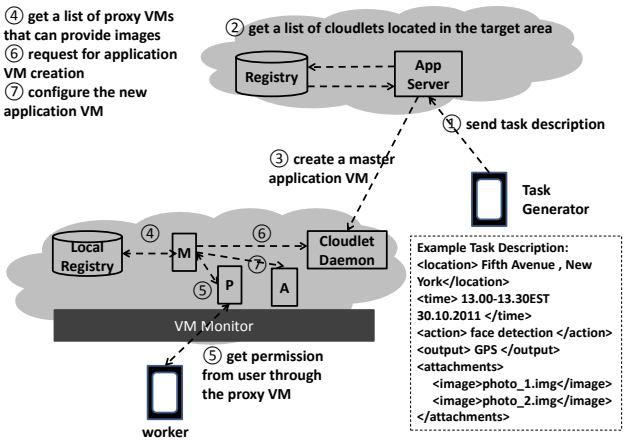

Figure 2: Workflow of crowd bootstrap.

1. The task generator (here, it is the parents' smartphone) constructs and sends a task description to the application server, typically located in the public cloud. The actual format of this can be application-specific, but is shown as an XML snippet here. The critical information includes type of search (face detection), the sample images, and a location area to scope the search. How this description is constructed and communicated is also left to the application, e.g., with a front-end app on the phone, or through a web-form on the application server.

2. The application server parses the task description, and consults a global registry for a list of cloudlets that are located in the target area.

3. The application server contacts the cloudlet daemon on each target-area cloudlet, and requests a MAVM instance be created. It forwards the VM disk image and memory snapshot to launch the MAVM. In practice, techniques employing demand paging or VM synthesis can minimize overheads of launching the MAVM.

4. The MAVM on each target cloudlet uses a cloudlet-local registry to discover proxy VMs connected to devices that can provide the desired sensor data (here, videos and images).

5. The MAVM requests participation from the mobile users through the proxy VMs. Depending on user-defined policies, the proxy may require explicit permission from the user, or the proxy VM can automatically join crowds on behalf of the user when particular criteria are met (e.g., share video when in a public space, but not audio).

6. Once permission is granted, the MAVM will request the cloudlet daemon to create application VMs. In practice, these can simply be clones of the MAVM, operating in a different mode.

7. The MAVM configures the networking setup of the application VMs, while the the proxy VM will add the new application VM to the subscriber list.

When the above steps are finished, the proxy VMs will start forwarding images and videos to the application VMs, which will apply face detection and forward potential matches through the MAVM and application server to the parents' smartphone. We believe our architecture has the potential to bootstrap large crowds in just a matter of minutes, making this on-demand crowdsensing use case possible.

3.3 Benefits of Our Design

Our deployment model is architected to support scalable, efficient data sharing between multiple applications and users, while reducing the burden on application developers and end users. It scales up crowdsensing tasks by making it easier to access data from a larger pool of diverse smartphones, allow users to simultaneously particpate in multiple applications, and support rich, high-data-rate sensors at global scale.

Separating the process of data collection and sharing from application-specific processing, our system lets developers focus on the latter, rather than porting their application to a myriad of mobile platforms and understanding the idiosynchrasies of each. In fact, our deployment model increases the choices in programming languages, as the application is self-contained in its application VM and does not have to meet specific compatibility constraints for mobile platforms. Similarly, the developer is free to use a variety of programming models to distribute computation and aggregate results, and not forced to use a one-size-fits-all paradigm. Deploying VMs to users boils down to rapid cloning of the application VM on the cloudlet, regardless of the mobile hardware of the users. Best practices for privacy preservation and user-directed policies for participation can be implemented in the proxy VMs, and be applied to all crowdsensing applications. Our framework also allows flexibility in partitioning work between the proxy VM and mobile device. For example, supporting multiple applications with differing fidelity or resolution requirements simultaneously will entail some amount of preprocessing; this can be done in the proxy VM, mobile device, or a combination of both depending on hardware capabilities, processing overheads, and energy availability.

Users no longer need to install individual apps on their devices to participate in crowd sensing. Rather, they need only grant permission, and if willing, can direct their proxies to automatically participate in some forms of crowdsensing. This allows for very rapid deployment of crowd applications. Demands on the mobile device can also be reduced, as processing is offloaded to the cloud, and only a single copy of the sensor data is uploaded even when participating in multiple applications. When a user leaves a crowd, the application VM is simply destroyed, and does not require additional attention from the user.

Lastly, our design performs processing and data aggregation close to the data sources. This brings two benefits: 1) it reduces traffic on wide-area networks; 2) it reduces network latency by avoiding long-distance data transmission through the backbone networks. This makes it possible to scale up crowdsensing with high-data-rate sensors. VM migration can ensure that processing remains close to data source even as users move around.

4. CHALLENGES

There are several technical hurdles in the path of a real-world deployment of our proposed architecture. We discuss these below.

4.1 Virtualization Overhead

Leveraging virtualization allows us to create a flexible platform in a multi-party setting where user privacy, scalability and isolation between individual crowdsensing applications are key requirements. These advantages come at the price of both VM creation overhead and the need for more advanced inter-VM communication management. In our design, a new clone of the application VM is instantiated for each user joining the crowd. Ideally, this new VM should start as fast as possible with minimal cost on resources. In practice, when a VM Monitor starts a new VM, it must first reserve all of the memory resources needed for the VM. This constraint prevents rapid creation of multiple VMs concurrently.

One way to solve this problem is to reduce the number of running VMs by replacing the per-user application VMs with one multiplexing application VM on each cloudlet. However, this will introduce the complexity of process migration in mobile scenario when any hard state contained in the application VMs must be preserved. An alternative way is to reduce the overhead of VM creation through advanced cloning mechanisms. There are several works that try to reduce the memory copy overhead by cloning the memory from running VMs. SnowFlock [11] proposes to fetch memory on demand while cloning VMs. It manages to clone 32 clones in 32 different hosts within one second by combining on-demand fetching with TCP multicasting for network scalability. Kaleidoscope [5] takes this one step further by discriminating VM memory state into semantically related regions to achieve prefetching and efficient transmitting.

An additional challenge is configuration and performance of inter-VM communication. The performance of inter-VM communication is relatively low compared to inter-process communication. When the system workload on the cloudlet increases, this may result in delayed transmission of sensor data between proxy and application VMs. Note that this low performance is due to inefficient CPU scheduling of the host, as the physical network interface is not touched by inter-VM traffic.

4.2 Migration-induced Reconfiguration

Physical mobility of a device may trigger the migration of the proxy VM and the associated application VMs that are not stateless. Consequently, the IP address of the mobile device as well as those of the VMs may change. To maintain established connections between mobile device and proxy VM, as well as between proxy VMs and application VMs, automated advanced network reconfiguration is needed. This potentially includes network addressing, NAT settings and firewall setup in VMs. Due to this overhead, IP-based solutions may not provide adequate performance in our envisaged scenarios. Non IP-based solutions such as the Host Identify Protocol [16] have been designed from scratch with these limitations in mind, but these protocols still need to be evaluated in real networks. The deployment of these is unlikely to be easy, given the fact that today's Internet is built almost exclusively on the TCP/IP stack.

4.3 Standardization of Sensing Interfaces

Sensor data is distributed from the proxy VMs to the application VMs through a publish-subscribe mechanism. Standard sensor data descriptions are needed to realize communication between proxy VMs and application VMs of various developers. As discussed in Section 3.3, some efforts [29, 21] have been invested on developing such interfaces, however, unfortunately so far no consensus has been made yet.

Another challenge lies in the fact that different crowdsensing applications might be built on the same sensor data, but require a different format or sample rate. However, the sensor data collected from the devices provided by different vendors may not be able to always provide the data in the right format or at the right sample rate.

There is a trade-off to be studied on whether the conversion from the original sensor data to the requested output format(s) must be done on the mobile device, the proxy VM or inside the application VM itself. At first sight, running inside the application VM is the most logical choice, as it removes as much logic as possible from the mobile device and the proxy VM. However, this results in a lack of synchronization and a potential waste of resources. For example, what if all currently running application VMs only need camera frames at 10 fps, while the mobile device emits at a standard 30 fps? In this case, it would make sense to put downsampling application logic on the mobile device, and to put logic in the proxy VM that can configure the sensor capturing on the mobile device. When a new application VM is deployed needing 15 fps, the proxy VM may instruct the mobile device to increase its frame rate accordingly. Support for device-level configuration may vary significantly by platform and specific sensor hardware, so proxy VMs need to be designed to abstract away such differences.

5. CONCLUSIONS

This paper has argued that the existing deployment model for crowdsensing applications does not support either efficient crowd scaling over heterogeneous mobile platforms or the data sharing between crowdsensing applications. While VM-based cloudlets have been widely studied and utilized for computation offloading, we explore the potential uses of VM-based cloudlets for lowering the barriers to scaling up crowdsensing applications. Our solution leverages the existing programming frameworks for crowdsensing applications. There are still several challenges that must be addressed before this kind of deployment model can be adopted, we are currently implementing the deployment platform with specific focus on the research challenges discussed in this paper.

6. REFERENCES

[1] P. Bahl, M. Philipose, and L. Zhong. Vision: cloud-powered sight for all: showing the cloud what you see. In *Proc. of the third ACM workshop on Mobile cloud computing and services*, 2012.

[2] R. K. Balan, K. X. Nguyen, and L. Jiang. Real-time trip information service for a large taxi fleet. In *Proc. of the 9th Intl. Conf. on Mobile systems, applications, and services*, 2011.

[3] X. Bao and R. Roy Choudhury. Movi: mobile phone based video highlights via collaborative sensing. In *Proc. of the 8th Intl. Conf. on Mobile systems, applications, and services*, 2010.

[4] A. Bozzon, M. Brambilla, and S. Ceri. Answering search queries with crowdsearcher. In *Proc. of the 21st Intl. Conf. on World Wide Web*, 2012.

[5] R. Bryant, A. Tumanov, O. Irzak, A. Scannell, K. Joshi, M. Hiltunen, A. Lagar-Cavilla, and E. de Lara. Kaleidoscope: cloud micro-elasticity via vm state coloring. In *Proc. of the sixth Conf. on Computer systems*, 2011.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proc. of the 8th Intl. Conf. on Mobile systems, applications, and services*, 2010.

[7] T. Fujisaka, R. Lee, and K. Sumiya. Discovery of user behavior patterns from geo-tagged micro-blogs. In *Proc. of the 4th Intl. Conf. on Uniquitous Information Management and Communication*, 2010.

[8] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11), 2011.

[9] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proc. IEEE*, 2012.

[10] E. Koukoumidis, L.-S. Peh, and M. R. Martonosi. Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory. In *Proc. of the 9th Intl. Conf. on Mobile systems, applications, and services*, 2011.

[11] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: rapid virtual machine cloning for cloud computing. In *Proc. of the 4th ACM European Conf. on Computer systems*, 2009.

[12] N. D. Lane, Y. Xu, H. Lu, S. Hu, T. Choudhury, A. T. Campbell, and F. Zhao. Enabling large-scale human activity inference on smartphones using community similarity networks (csn). In *Proc. of the 13th Intl. Conf. on Ubiquitous computing*, 2011.

[13] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell. Soundsense: scalable sound sensing for people-centric applications on mobile phones. In *Proc. of the 7th Intl. Conf. on Mobile systems, applications, and services*, 2009.

[14] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrasekaran, W. Xue, M. Gruteser, and W. Trappe. Parknet: drive-by sensing of road-side parking statistics. In *Proc. of the 8th Intl. Conf. on Mobile systems, applications, and services*, 2010.

[15] MYOONET. Unique scalable data centers. `http://www.myoonet.com/unique.html`, 2011.

[16] P. Nikander, A. Gurtov, and T. Henderson. Host identity protocol (hip): Connectivity, mobility, multi-homing, security, and privacy over ipv4 and ipv6 networks. *Communications Surveys Tutorials, IEEE*, 12(2):186 –204, 2010.

[17] Z. Niu, S. Li, and N. Pousaeid. Road extraction using smart phones gps. In *Proc. of the 2nd Intl. Conf. on Computing for Geospatial Research & Applications*, 2011.

[18] PC World. `http://www.pcworld.com/article/255519/verizon_to_offer_100g_links_resilient_mesh_on_optical_networks.html`, 2012.

[19] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan. Medusa: a programming framework for crowd-sensing applications. In *Proc. of the 10th Intl. Conf. on Mobile systems, applications, and services*, 2012.

[20] M. Rabbi, S. Ali, T. Choudhury, and E. Berke. Passive and in-situ assessment of mental and physical well-being using mobile sensors. In *Proc. of the 13th Intl. Conf. on Ubiquitous computing*, 2011.

[21] L. Ravindranath, A. Thiagarajan, H. Balakrishnan, and S. Madden. Code in the air: simplifying sensing and coordination tasks on smartphones. In *Proc. of the Twelfth Workshop on Mobile Computing Systems and Applications*, 2012.

[22] M. Satyanarayanan. Mobile computing: the next decade. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing and Services*, July 2010.

[23] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4), Oct. 2009.

[24] S. Shah, F. Bao, C.-T. Lu, and I.-R. Chen. Crowdsafe: crowd sourcing of crime incidents and safe routing on mobile devices. In *Proc. of the 19th ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, 2011.

[25] A. Tang and S. Boring. #epicplay: crowd-sourcing sports video highlights. In *Proc. of the 2012 ACM annual Conf. on Human Factors in Computing Systems*, 2012.

[26] S. Wakamiya, R. Lee, and K. Sumiya. Crowd-sourced urban life monitoring: urban area characterization based crowd behavioral patterns from twitter. In *Proc. of the 6th IIntl. Conf. on Ubiquitous Information Management and Communication*, 2012.

[27] M. Wirz, C. Strohrmann, R. Patscheider, F. Hilti, B. Gahr, F. Hess, D. Roggen, and G. Tröster. Real-time detection and recommendation of thermal spots by sensing collective behaviors in paragliding. In *Proc. of 1st Intl. symposium on From digital footprints to social and community intelligence*, 2011.

[28] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proc. of the 8th Intl. Conf. on Mobile systems, applications, and services*, 2010.

[29] F. Ye, R. Ganti, R. Dimaghani, K. Grueneberg, and S. Calo. Meca: mobile edge capture and analysis middleware for social sensing applications. In *Proc. of the 21st Intl. Conf. companion on World Wide Web*, 2012.

[30] YouTube. `http://www.youtube.com/t/press_statistics`, 2012.

[31] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proc. of the 18th Intl. Conf. on World wide web*, WWW '09, 2009.

[32] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of the 10th Intl. Conf. on Mobile systems, applications, and services*, 2012.

[33] J. Zimmerman, A. Tomasic, C. Garrod, D. Yoo, C. Hiruncharoenvate, R. Aziz, N. R. Thiruvengadam, Y. Huang, and A. Steinfeld. Field trial of tiramisu: crowd-sourcing bus arrival times to spur co-design. In *Proc. of the 2011 annual Conf. on Human factors in computing systems*, 2011.