vSensor: Toward Sensor-rich Mobile Applications

Minsung Jang, HyunJong Lee, Ketan Bhardwaj, and Karsten Schwan
School of Computer Science
Georgia Institute of Technology

ABSTRACT

Interacting with sensors presents numerous challenges, particularly for applications running on resource-constrained platforms like smartphones or tablets. This paper suggests the vSensor abstractions that provide such applications with ease of use for today's multitude of sensors. vSensors not only assist with sensor use, but also interact with nearby and cloud resources to make available to applications the resources required for processing sensor-acquired data and controlling actuators, thus permitting them to scale in sensor usage. The vSensor abstraction's Android implementation offers globally transparent, uniform access to the ambient sensors in the environment in which apps operate, supporting both unmodified apps and those with vSensor awareness.

I. Introduction

The Gartner group predicts the total number of sensors to grow to 26 billion units worldwide by 2020 [1]. This plethora of sensors enables a rich set of applications that can assist users in their daily lives, to help capture user intent and context [2], to enable intuitive and meaningful user interaction, and to assist with specific tasks as in elder care [3], efficient driving [4], etc.

Our work and this paper address the challenges faced by applications that seek to leverage and use the dynamic sets of sensors present on mobile devices and in the environments in which they operate, including:

- the diverse nature of those sensors, in part because of the fragmented nature of today's sensor ecosystem;
- the computational and data management challenges in interacting with those sensors, particularly for applications running on resource-constrained end devices like smartphones or tablets; and
- the dynamic nature of sensor presence, as users move in and out of their proximity and run applications requiring their dynamic access.

A simple example of the latter is a smartphone app desiring to interact with sensors in the home when the homeowner pulls into the garage. Doing so requires the app to dynamically acquire access privileges to those sensors, followed by subsequent phone-sensor interactions that involve extensive communication and processing activities, and developing such app functionality involves understanding and handling the rich variety of sensors present in the home and finding ways to run app functions within the constrained resources available on smartphones.

The vSensor (virtual sensor) abstraction developed in our research makes it easier for applications to leverage and exploit the on- and off-device sensors present in their current environments, particularly for apps running on resource-constrained devices like smartphones. vSensors

- *virtualize* individual sensors – virtual sensors – to obtain a uniform way to interact with them, thus removing from applications the need to understand in detail how to interface with the many specialized and custom sensors that surround them; and
- *externale* sensor interactions and processing from the resource-constrained mobile device to nearby and remote cloud resources – vSensor abstractions – to leverage the computational and storage abilities for running the complex functionality implemented by sensor-rich applications.

Figure 1 depicts how vSensors make available to applications the sensors present in their current environments, with application services running on both edge and datacenter cloud systems, ranging end user devices, to nearby computing resources (e.g., a home PC), and/or in the remote cloud.

vSensors are implemented for Android platforms, and they leverage the representative 'edge cloud' infrastructure described in [5], the latter permitting their services to run 'anywhere', i.e., on devices, nearby machines, and the remote cloud. vSensors extends these underlying cloud functionalities with the sensor abstraction and with access control methods: sensor-based applications can construct virtual sensors, create services that run on edge cloud (i.e., nearby) or remote cloud resources, and orchestrate their execution as sets of services running on the device and/or in the cloud.

This paper's evaluations of vSensor with the prototype applications show them fully interacting with a variety of ambient sensors and resources, without additional overheads compared to the less convenient, direct application-sensor interactions, with the additional ability to scale those interactions to 100s of sensors, actuators, and associated services, even on resource-constrained platforms like smartphones.

The remainder of the paper is organized as follows. Mobile applications' (apps') current usage of sensors is explained in Section II, with vSensor-enabled use cases shown in Section III. The design and implementation of vSensor are described in Sections IV and V, respectively. Experimental results in Section VI demonstrate our initial implementation results. Conclusions appear in Section VII.

II. Motivation

With the advent of a world of a trillion sensors and mobile devices, a new class of applications has been predicted to emerge rapidly, providing end users with personalized services

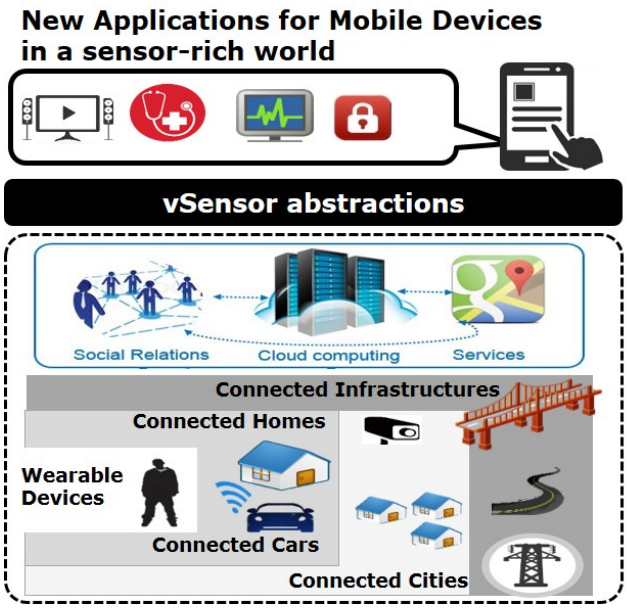


Fig. 1: Vision of vSensor in a Sensor-rich world

based on the precise capture of their current contexts and intents. Doing so, however, requires those apps to interact with the numerous sensors present on mobile devices and in users' current environments, to extract personal user contexts and environmental conditions. Unfortunately, today's reality is that most mobile applications (apps) hardly use sensors, despite the fact that the devices hosting such apps are themselves sensor-rich. Estimates [6] are that apps using at least one sensor in Android devices are just 0.5% of all available apps in 2012, with typical devices equipped with six sensors on average.

This section updates such estimates with a comprehensive study of how today's mobile applications interact with sensors. The intent is to better understand how apps interact with sensors, to motivate the functionality of vSensor intended to benefit them. The study is also a starting point for how vSensor functionality can enable new classes of mobile apps.

A. Methodology

Our study uses a tool that downloads apps from the Google Play Store that meet some predefined conditions, based on recently announced techniques like [7]. For such apps, we then analyze how they behave on the Android platform by scrutinizing their Dex bytecodes and manifest files. An initial study used the top 100 apps in each category on the Google Play Store, resulting in a total of 5000 apps (on August 21, 2014). We plan to expand to almost all free apps in the Store.

B. Discussion of Study Outcomes

As shown in Table I, for the top 100 apps, 81 out of 5000 (1.62%) use at least one sensor. This constitutes only a small increase over the previous estiate of 0.5% reported in 2012.

Evident from the statistics reported above is the fallacy of predictions made in the academic literature and in industry that mobile devices will become hubs in a sensor-rich world.

Sensor Count	Apps(top 100)
1	64
2	13
3	3
4	0
5	1
6	0
7	0
8	0
Total(%)	81 (1.62%)

TABLE I: The number of apps based on their sensor use. The total number of top 100 apps in each category of the Google Play Store are 5000 on August 21, 2014.

In theory, many applications could benefit from leveraging all sensors on devices and in a user's current environment (e.g., sensors from wearable devices, homes, and cars). In practice, this is not the case, and it remains unclear why today's apps do not actively leverage even the potential utility of the sensors present on their own devices.

vSensor's approach in reaction to these results is to try to make it easier for apps to use on-device and nearby sensors, (i) by logically decoupling such physical sensors to meaningful sensor abstractions (see Section IV-A), and (ii) by providing a flexible service execution for processing sensor data (see Section IV-A).

III. Applications with vSensor– Current and Future Use Cases

This section describes sensor-rich applications written with vSensor, able to easily leverage surrounding sensors to collect notable events and employ nearby resources to process or/and react to them.

A. Prototype Applications with vSensor

1) Unmodified Store Apps using Local and Remote Sensors: A goal of vSensor is to make it easier for existing applications to transparently and seamlessly scale in terms of numbers of sensors with their associated data processing activities, without the need to modify said applications whenever additional or new sensors are used. We demonstrate this functionality with a simple Play Store applications, Sensor Readout, which reads the outputs produced by all sensors embedded in an Android device. With vSensor, an unmodified app like this can interact with both on- and off-device accessible sensors.

2) Composed Sensors – Combining Sensors via Services: vSensor can abstract from individual physical sensors to form new sensor types, but a common application need is to combine and make use of multiple sensors to realize some desired app-level functionality. We consider such functionality another case of sensor virtualization and extension. vSensors can group many sensors of the same types, e.g., all of the light sensors in a home. A concrete example evaluated in our work is an application permitting end users to check the current time (on their smartphone), but without turning on the smartphone's battery-consuming screen. This 'don't turn on the screen' application uses a set of virtual sensors offered by vSensor with access to a home camera and a phone's speaker, and it runs its some complex software services on the PCloud.

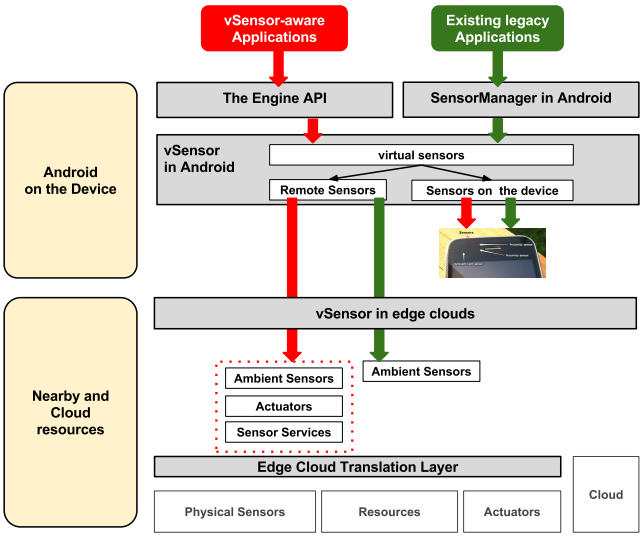


Fig. 2: vSensor Design

The service implements finger-gesture recognition using the camera: if the user approaches the smartphone with two fingers, this is interpreted as a desire to check time rather than grasp the phone;the response is the phone's speaker stating the current time, without unlocking and activating the screen, thus conserving phone power.

B. Future Applications

An additional application currently under development realizes the aforementioned 'health virtual sensor'. This application will allow mobile devices to become hubs for health-related information about the device owner. The application periodically measures data like blood sugar levels, data acquired from wearable devices (e.g., *health bracelets*), and even data acquired from say, an exercise bike used by the owner in a health club. Analytic services part of the object immediately raise alarms if unusual readings are detected from the object's virtual sensors, and in addition, they interact with a cloud-resident service to compute long term health statistics, implement a dashboard, etc.

IV. vSensor Design

vSensor's key goal is to enable apps running on smartphones and similarly resource-constrained end user devices to more easily and efficiently interact with the many sensors present on the devices themselves and in their surroundings. Concerning ease of use, vSensor seeks to overcome difficulties related to using many diverse sensors and to handle their potential incompatibilities and custom APIs. Concerning efficiency, vSensor makes possible the use of ambient nearby resources and of the remote cloud, for complex processing and transformation of sensor data. Figure 2 depicts an overview of vSensor's design, discussed in more detail next.

A. Design Principles

The vSensor abstraction enables apps to create points of access and use for sensors and associated sensor services and actuators, for single or sets of sensors and regardless of where these sensors are physically located, i.e., on the app's local platform or accessible remotely.

Logical decoupling – sensor virtualization. vSensor virtualizes physical sensors, the goal being to create meaningful sensor abstractions with uniform APIs vs. the diverse, custom APIs of available physical sensors. Virtualization also provides substantial flexibility for crafting exactly the abstractions desired by applications, an example being a 'health' sensor implemented by virtual sensors comprised of a scale, a heartbeat sensor on a bracelet, and cameras watching the exercise equipment being used.

Flexible service execution – sensor services. The location of a virtual sensor should not determine where its service codes manipulating sensor data are run: on the sensor itself, on the platform hosting it, or on other available processing resources. There is ample previous work demonstrating the need for such flexibility, ranging from early work on adaptive sensor processing [8] or adaptive media manipulation [9] to recent work on edge cloud functionality [10] and cloud offloading for sensors [11]. In accordance with such insights, virtual sensors are able to run on any underlying platform made available by edge cloud infrastructures [5], [12], [13], [14] with which vSensor can interact (see the 'translation layer' in Figure 2). Currently, this is the PCloud edge cloud [5], but it is straightforward to write such a layer for other infrastructures.

Backward compatibility – sensor illusions. The physical sensors wrapped via vSensor abstractions should remain available to existing applications that use them. We achieve this backward compatibility by having vSensors export 'illusions' of those sensors to legacy apps, while at the same time, providing the advanced functionality of use to new applications via a new set of our APIs.

B. Basic Operations

Virtual sensors are created and managed with operations that include *group, glance, aggregate, trim, bundle,* and *reconfigure* to meet the design principles articulated in Section IV-A. Apps invoke such operations via the 'Engine API' on the Android platform, also shown in Figure 2 and as with object services, these operations can again be run on any of the resources made available by the underlying edge cloud – on the device making the call, on some nearby resource, or in the remote cloud.

V. Select Implementation Detail

This section presents the implementation detail needed for understanding the performance evaluation in Section VI.

A. Android Realization

The vSensor Android implementation must be (i) backwards compatible, (ii) transparent to their physical nature, and (iii) portable, to permit virtual sensors to run on any of the variety of edge cloud infrastructures currently under development [12], [13], [5]. We obtain these properties as follows. First, to existing sensor-based apps, virtual sensors wrapping existing sensors provide to these apps the aforementioned 'illusions' of the physical sensors these apps currently employ.

Name	Hardware/Role(s)
Sensor Nodes	Intel Galileo 1st Gen
Actuator Nodes	Intel Galileo 1st Gen
Camera Nodes	Exynos 5420 and AMD E450
EC2	m3.large (Cloud resource)
Local 1	Intel i5 (Local resource 1)
Device	Galaxy S4 (User's mobile device)

TABLE II: vSensor Testbed Setting

Second, with sensor virtualization and the ability to construct entirely new sensors from both physical sensors and events (e.g., calendar events) associated with logical artifacts, transparency becomes a pervasive property of the vSensor implementation. Third, we obtain portability to different edge cloud infrastructures by providing a translation layer for running vSensors on the PCloud edge cloud.

B. The Engine API

In Android, apps access virtual sensors via the 'Engine API', which implements methods that correspond to the operations described in Section IV-B. The API provides access to both local and remote sensors, via virtual sensors wrapped by the vSensor abstractions.

VI. Experimental Evaluation

We evaluate the prototype applications described in Section III-A to demonstrate the utility and performance of the vSensor abstraction.

A. Evaluation Setup

Table II describes a set of resources in our experiments. Intel Galileo boards(1st Generation) are used for remote sensor and actuator nodes. The remote sensor nodes measure humidity and temperature at their locations while the actuator nodes send notifications to users via the embedded LED. The camera nodes keep track of the figure's gesture of users.

B. Initial Results

1) Unmodified Store Apps using Local and Remote Sensors: We verify the backward compatibility of the vSensor abstractions by checking whether the existing apps in the Google Play Store can transparently access the physical sensors they already use via vSensor-based virtualized, an implicit bonus of such compatibility being that such sensors can be local or remote. Figure 3 is a screenshot showing that such an app, Sensor Readout, transparently interacts with local (on-device) and remote (off-device) sensors. First three sensors in Figure 3 are those on our test device, Samsung Galaxy S4, while the rest are virtual sensors that are local illusions of remote sensors created by the vSensor abstraction.

2) Composed Sensors – Combining Sensors via Services: The Engine API enables mobile applications to combine different virtual sensors into a new single virtual sensor that in turn interacts with their respective actuators and services. The 'don't turn on the screen' application demonstrates this capability by combining a set of cameras at home into a single virtual sensor acting as a 'user's finger gesture detection

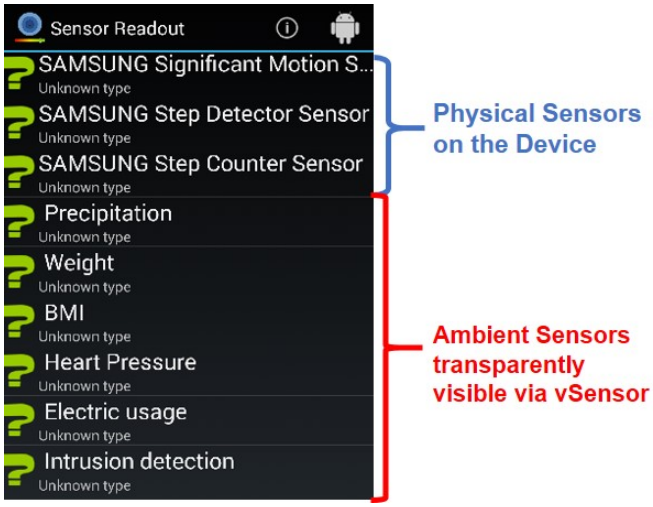


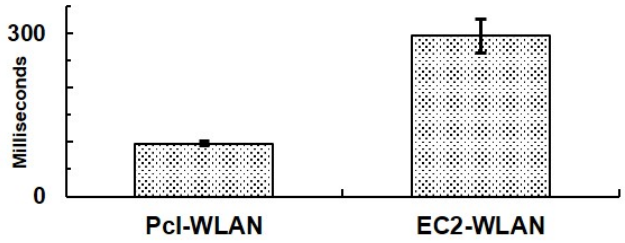
Fig. 3: A Screenshot of the Sensor Readout App

sensor'. The application uses this virtual sensor and runs its finger-gesture recognition service on nearby resources or on the remote cloud (Amazon EC2), based on a scheduling decision made by the underlying edge cloud infrastructure, PCloud in our case. The end user's desire – a reading of the current time – is reported via the phone's speaker. Results in Figure 4 summarize our experiments for the app. We check the current time ten times during ten seconds by either turning on the screen (labeled as 'Screen') or via the app (labeled as 'Speaker'). Figure 4 (a) shows the elapsed time from when the finger detection service is run and the current time is stated, with variations due to the fact that the recognition service can run in different locations – on local edge cloud devices or in the remote cloud. If the service runs on local resources (labeled as 'Pcl-WLAN'), the response time is around 97.4ms, while for the remote cloud (labeled as 'EC2-WLAN'), it is 294.5ms. We also compare the power and energy consumption of the device running this app vs. simply activating the screen and permitting the user to see the time. Figure 4 (b) clearly shows that using this app can dramatically reduce the device's energy consumption, by up to 46%.

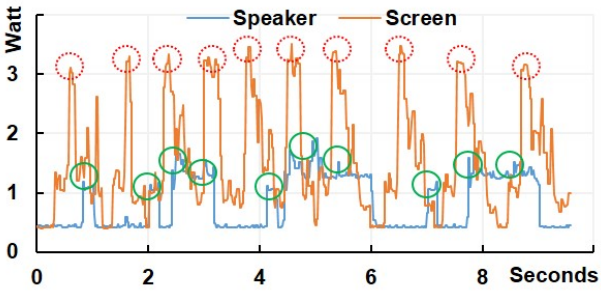
These initial results for two prototype applications shows that (i) existing apps can interact not only with sensors on the devices where they run, but also with those available remotely, without having to change their source code, and that (ii) vSensors can help open up new opportunities for mobile applications to use sensors in innovative ways, including via sensor composition and the ability to run potentially complex processing methods on resources other than those available on a single device.

VII. Conclusions and Future Work

With the vSensor approach and abstractions, we address several issues with today's sensors and sensor processing ecosystem. First, with their uniformity, virtual sensors make it easier for applications to interact with and leverage available custom and specialized sensors. Second, by leveraging edge and remote clouds, vSensor makes feasible complex sensor processing and integration activities that are not limited by a smartphone's computing, storage, and battery constraints.



(a) Response time to state the current time. Error bars show 95% confidence intervals, repectively.



(b) Power consumption

Fig. 4: Results of the 'Don't turn on the screen' application. Concerning total energy consumption, the speaker and the screen consume 22.14 and 40.94 mWH, respectively. Each circle shows the moment that a user acknowledges the current time.

The vSensor abstractions help applications scale to using a rich variety of sensors and sensor data belonging to different owners. Since such usage may rise concerns regarding privacy risk due to the nature of sensors, vSensor should not allow the applications to indiscreetly access those sensors. Hence, third, we will expand vSenor to provide access control methods for sensor owners, who will be able to regulate application accesses via defined access rights, e.g., to preserve privacy. Further, we will lessen the burden placed on owners by the need to explicitly determine access controls via automation methods leveraging owners' social relationships and the contexts in which accesses are made.

REFERENCES

[1] "Press release," http://goo.gl/CziKJa.

[2] K. Bhardwaj, S. Sreepathy, A. Gavrilovska, and K. Schwan, "Ecc: Edge cloud composites," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, April 2014, pp. 38–47.

[3] L. C. D. Silva, C. Morikawa, and I. M. Petra, "State of the art of smart homes," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 7, pp. 1313 – 1321, 2012, advanced issues in Artificial Intelligence and Pattern Recognition for Intelligent Surveillance System in Smart Home Environment. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095219761200098X

[4] S. Thrun, "Toward robotic cars," *Commun. ACM*, vol. 53, no. 4, pp. 99–106, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721679

[5] M. Jang, K. Schwan, K. Bhardwaj, A. Gavrilovska, and A. Avasthi, "Personal clouds: Sharing and integrating networked resources to enhance end user experiences," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 2220–2228.

[6] "Sensor-based apps offer lots of potential but are hindered by fragmentation," http://goo.gl/aUhi8N.

[7] "Dex2jar tool," https://code.google.com/p/dex2jar/.

[8] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha, "On adaptive resource allocation for complex real-time applications," in *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, Dec 1997, pp. 320–329.

[9] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 145–158. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296820

[10] M. Buevich, A. Wright, R. Sargent, and A. Rowe, "Respawn: A distributed multi-resolution time-series datastore," in *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, Dec 2013, pp. 288–297.

[11] B. Liu, Y. Jiang, F. Sha, and R. Govindan, "Cloud-enabled privacy-preserving collaborative learning for mobile sensing," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '12. New York, NY, USA: ACM, 2012, pp. 57–70. [Online]. Available: http://doi.acm.org/10.1145/2426656.2426663

[12] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 25–25. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228332

[13] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14. New York, NY, USA: ACM, 2014, pp. 68–81. [Online]. Available: http://doi.acm.org/10.1145/2594368.2594383

[14] "Smartthings," https://www.smartthings.com.