Towards a Rich Sensing Stack for IoT Devices

Chenguang Shen, Haksoo Choi, Supriyo Chakraborty, Mani Srivastava
Networked and Embedded Systems Lab
University of California, Los Angeles
Los Angeles, CA, USA
{cgshen, haksoo, supriyo, mbs}@ucla.edu

***Abstract*—The broad spectrum of interconnected sensors and actuators, available on various mobile devices and smartphones and collectively defined as the Internet of Things (IoT), have evolved into platforms with ability to both collect personal sensory data and also change the users' immediate environment. The continuous streams of richly annotated sensory data on these IoT devices have also enabled the emergence of a new class of context-aware apps that use the data to infer user context and accordingly customize their responses in real-time. However, this growth in the number of apps has not been complemented with adequate system support on the IoT devices resulting in monolithic apps that each implement and execute their own customized sensing pipelines. In this paper, we outline our vision of a sensing stack, akin to a networking stack, that can facilitate the development and execution of context-aware apps on IoT devices. There are several advantages to building a rich sensing stack. First, it allows apps to reuse stages of the sensing pipeline easing their development. Second, the layers of the stack allow for both in- and cross-layer resource optimization. Finally, it allows better control over the shared data as instead of raw-sensor data, higher-level semantic abstractions, such as inferences can now be shared with apps. We describe our initial efforts towards creating the different building blocks of such a sensing stack.**

Keywords—sensing, context inference, the Internet of Things

I. INTRODUCTION

In recent years, we have seen the emergence of networked embedded devices such as smartphones, tablets, smart watches/glasses, intelligent building devices, and even smart vehicles. These devices, collectively forming the Internet of Things (IoT) and, owing to their ubiquity, have become continuous sensors of human spaces. The continuous stream of richly annotated, real-time personal data made available by tapping into the spectrum of sensors available on these devices has further led to the emergence of a new class of context-aware apps. These apps use the data to infer diverse contexts and user behaviors, including transportation mode [4], conversation episodes [6], stress [7], and emotion [8] and provide customized services.

This growth in the number of apps has however not translated into adequate system support for app development on the IoT devices resulting in monolithic apps that each implement and execute their own customized sensing pipelines. The sensing APIs currently available on IoT platforms provide only basic functionalities, compared to the rich network stacks in mainstream mobile operating systems such as iOS and Android. For instance, to perform network communications, app developers can either use socket to send and receive raw data over transport layer protocols, e.g., TCP and UDP, or incorporate high level primitives, e.g., HTTP requests and responses. At the hardware level, the operating systems provide abstractions for the different radios such as Wi-Fi, cellular network, Bluetooth, or ZigBee, hiding complexity from the developers. In contrast, typical context-aware apps today directly access raw sensor data due to the lack of system support. App developers have to collect and buffer the data, implement various feature computation blocks, train a machine learning model, and wire all feature and classification computation modules together to complete a working sensing pipeline. Recently researchers have proposed solutions [9][10] to support the building of context-aware apps, but they either require a different programming paradigm from common app development or lack flexibility to configure sensing pipelines.

In this paper, we outline our vision of a sensing stack that can facilitate the development and execution of context-aware apps on IoT devices. Such a stack allows apps to reuse stages of the sensing pipeline simplifying the developer effort. In- and cross- layer optimization can be performed to better utilize the computation and communication capabilities of the IoT devices The stack has better control over shared data as instead of raw sensor data, higher-level semantic abstractions, such as inferences can now be shared with the apps. Our contributions are listed below:

- Using the example of a canonical sensing pipeline, typically implemented by context-aware apps, we outline the challenges towards the design and implementation of a sensing stack for the IoT.
- We identify the functionalities that should be provided by the sensing stack.
- We describe our initial efforts in creating the building blocks for the rich sensing stack, including exploiting processor heterogeneity for always-on low-power context inference, incorporating IoT devices in a Universal Device Service, efficient context inference execution using FlowEngine, and enforcing context-aware privacy on mobile operating systems through ipShield.

II. System Architecture

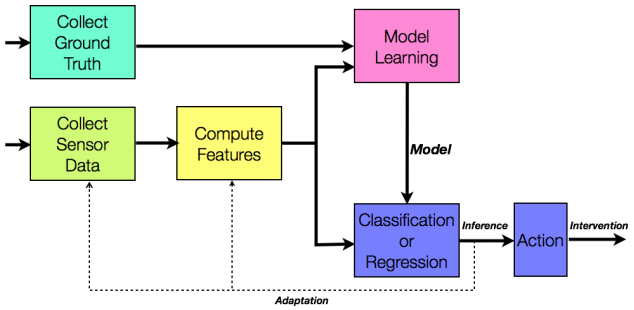

Fig. 1. Canonical Sensing Pipeline

Fig. 1 shows a canonical sensing pipeline commonly adopted by inferences in context-aware apps. Typically app running on the IoT devices subscribes to sensor data streams. Hardware fabric, device drivers, and system services work in concert to pass the sensor samples to the subscribing apps. There are mainly two phases in the lifetime of a context-aware app:

- **Learning**. The app first collects data and ground truth labels to perform the training. It extracts features to reduce the dimensionality of sensor data. The features, together with ground truth labels collected from the user, are used to learn a model for the sensing pipeline. In reality, context-aware inference apps are often shipped with a model previously learned, normally on the cloud, so the app is ready for classification after installation.
- **Classification**. The app collects new data and, with the model, it performs classification over the stream of extracted features to infer context labels (i.e., which class the current context falls under). The context label is later used as evidence for other decisions or actions.

However, due to the lack of proper sensing stack on IoT platforms, the development and execution of context-aware apps face myriad of technical challenges:

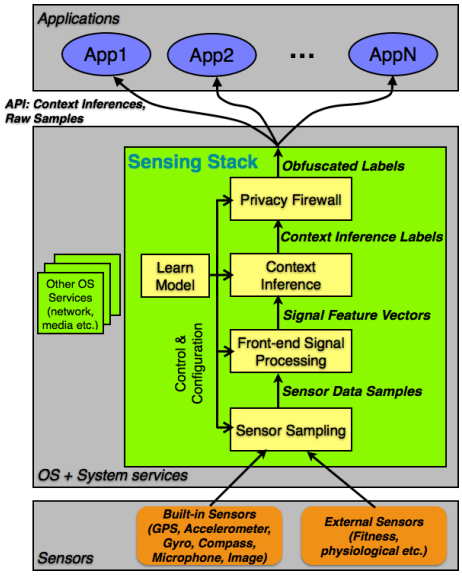

Fig. 2. Architecture of Sensing Stack

High energy consumption. The algorithms used in context-aware apps, including feature computation and classification, are computationally intensive and power hungry. Additionally, they often need to run in the background continuously to monitor user contexts and behaviors for just-in-time feedback, notifications, and interventions. For example, Kansal et al. [10] observed a reduction of phone standby time from 430 to 12 hours when an app is continuously using the proximity API in Android. The high power consumption of context-aware apps and the reduction in battery life as a result will only get worse as even more sophisticated context inferences emerge.

Lack of programming support. Currently there is only limited support provided by mobile operating systems for the development of context-aware apps, such as Apple's API suite in iOS for the M7 low-power motion co-processor [13], and Google's Play Services [14]. These APIs and services allow apps to subscribe to high-level phone contexts, but do not support personalization of sensing pipelines and the machine learning models used. The design and implementation of the sensing pipeline often requires domain expertise in signal processing and machine learning.

Information leakage. The majority of context inference apps available in IoT app ecosystems, such as Google's Play Store and Apple's AppStore, are published by third-party developers. However, not all app developers are equally trustworthy, and this fact coupled with user naiveté leads to data misuse and privacy concerns. Sensors, such as accelerometer, gyroscope, light, etc., are commonly considered to be innocuous. However, specific vulnerability of unrestricted access to accelerometer and gyroscope data has been exploited to mount keylogging attacks [5], and for reconstruction of travel trajectories [3].

This paper proposes the design of a rich sensing stack on IoT that addresses the above challenges. The architecture of our proposed sensing stack is shown in Fig. 2. The sensing stack features the following functionalities:

Provide inferences as high level abstraction: Instead of feeding raw sensor data from the hardware to subscribing apps, the sensing stack raises the abstraction and offers higher-level semantic abstractions, such as inferences to apps.

Rich context API: The sensing stack provides API to app developers for customizing the sensing pipeline and subscribing to specific inferences. This enables the reuse of computation modules and eases the app development. Feature vectors are calculated from sensor data, based on which the context inference module will compute the context labels and then provide results to subscribing apps.

Context inference as a shared system service: The sensing stack takes requests from all apps and performs context inference computations in a shared system service. The system service performs in- and cross- layer resource optimization and management to better utilize the computation and communication capabilities of the IoT devices

Enforcement of context-aware privacy: The sensing stack has better control over shared inference, compared to the case where raw sensor data is being shared, as the stack enables system level privacy enforcement on inferences. The privacy

firewall in the sensing stack generates privacy rules based on user inputs, and enforces the rules by obfuscating inferences.

Efficient execution of always-on context inference: The sensing stack leverages possible resources (e.g., heterogeneous cores) available on IoT devices for efficient always-on context inference in the background.

Fluid cooperation of IoT devices, smartphone, and the cloud: The sensing stack can not only leverage sensors on IoT devices, but also utilize sensors remotely connected to devices, e.g., via Bluetooth or Wi-Fi. The execution of context inference can be migrated across edge devices, smartphones, and the cloud for optimal performance and energy efficiency.

III. Towards a Rich Sensing Stack

In this section, we describe our effort in creating the building blocks towards a rich sensing stack from several perspectives.

A. Energy Efficient Context Inference

On current IoT devices, the always-on context inference apps run on the main app processor, i.e., the CPU (typically a high-end ARM processor core), and represent a significant portion of overall workload and power consumption. However, under the hood, mobile processor chips are not simply app processors. Rather, they are sophisticated system-on-chips (SoCs) where the main app processors are complemented by embedded processors, digital signal processors (DSPs) and graphic processing units (GPUs) that handle specialized work such as media processing and low-level sensor I/O. However, these auxiliary processors are usually hidden from the app programmer, and are instead limited to running prebuilt firmware provided by the platform manufacturer. Recently, conscious efforts are being undertaken by various mobile processor vendors to expose the co-processor heterogeneity to the app developers and provide them with the ability to program the DSPs. The Qualcomm Hexagon DSP is a representative example of such an embedded processor on mobile SoC. The recently released Qualcomm Snapdragon 800 series SoC [17] includes a Hexagon DSP with custom programmability and operates in the ultra-low power range. Likewise, the TI OMAP4 SoC [18] has a 500MHz C64x DSP and two low-end Cortex-M3 ARM cores on board.

In recent work [2], we explored the energy and performance implications of off-loading the computation associated with machine learning algorithms in context-aware apps, more specifically, the classification phase of context inference, to DSP cores. We have implemented two common classification algorithms, Support Vector Machine (SVM) and Gaussian Mixture Model (GMM), in DSP and measured their power consumption on both Pandaboard ES (TI OMAP4) [12] and MDP Tablet (Qualcomm Snapdragon 800) [11]. By profiling two apps, human activity recognition using accelerometer data and speaker recognition using microphone audio data, our results indicate that off-loading computation to the DSP reduces energy consumption by 17% and 60% for the two boards respectively, with negligible effect on app latency. To the best of our knowledge, this is the first work leveraging a powerful co-processor for context inference on mobile devices.

B. Incorporating IoT Devices

Mobile devices, due to their versatile capabilities, have become a key hub in the ecosystem of diverse IoT devices [16]. However, building context-aware apps coordinating with such IoT devices have become more challenging because they are built on top of many different communication technologies, including Bluetooth, Z-Wave, ZigBee, and Wi-Fi. Individual apps have to deal with such details. Even worse, each device manufacturer has their own communication APIs. Although several standardization efforts are going on but even those standards are competing with each other and the competition seems to be getting more crowded [15].

No matter how diverse the underlying technologies are, IoT apps must be independent of such technologies just like apps running on top of operating system's networking stack is unaware of the low-level details whether they are using Ethernet, cellular networks, Wi-Fi, etc. Therefore, we propose Universal Device Service, or UniDev, which hides underlying technical details of sensors and provides apps with uniform interface for communicating with diverse IoT devices. Sensors can be plugged into the UniDev framework via Bluetooth, Wi-Fi, Ethernet, etc. UniDev is a shared system service and has global knowledge of the device sharing across multiple apps. It is capable of optimizing device resource usage by coordinating different sensing rates of apps and sampling at optimal rate. In addition, UniDev provides a device discovery mechanism so apps can query currently available devices or get notified when the interested devices become available.

C. Execution of Sensing Dataflow

Inferring contexts from raw sensor data can be well modeled by dataflow graphs because context inferences typically consist of multiple series of steps such as data pre-processing, feature extractions, and classifications. In such dataflow graphs, computation nodes output their results to other nodes and/or take inputs from other nodes, forming an arbitrary graph structure. Therefore, our sensing stack includes FlowEngine, which adopts dataflow programming model for the execution of context inference workload. Adoption of dataflow programming model provides the following advantages in terms of programmability and resource usage:

FlowEngine provides a dataflow programming language, which eases the design of sensing dataflow with simple *declare* and *connect* syntax. It provides a library of dataflow nodes, and app developers just need to declare required nodes and then connect them to make a complete sensing dataflow. For example, calculating a root mean squared value from 3-axis accelerometer can be expressed as follows: `Acceleroemter acc; RootMeanSquare rms; acc -> rms;`

Without a shared system service like our sensing stack, individual apps have to directly process raw sensor data and implement their own context inferences. However, if multiple apps make the same inferences, scarce resources on smartphones are wasted due to redundant computation. Even if the multiple apps conduct different context inferences, they could still share common computation steps in the sensing dataflow graphs. For example, root mean squared values of 3-axis accelerometer readings could be used for both smartphone movement detection and transportation mode inference.

FlowEngine's dataflow execution model is able to identify common computation nodes and merge multiple dataflow graphs so that redundant computation can be avoided.

D. Context-aware Information Privacy

Smartphones and other mobile devices today are used to collect and share personal data with untrustworthy third-party apps, often leading to data misuse and privacy violations. As is discussed in Section II, unrestrictive access to the raw data from seemingly innocuous sensors can be exploited for severe privacy attacks. To protect users against unwanted privacy leakage from context inference apps, we present ipShield [1], a context-aware privacy enforcing framework for the Android operating systems. ipShield is open source and implemented by modifying the Android Open Source Project (AOSP).

ipShield allows users to specify their privacy preferences in terms of semantically meaningful inferences that can be drawn from the shared data. We took an important step towards presenting the privacy risks in a more *user-understandable* format. We modified the Android OS to monitor all the sensors accessed by an app regardless of whether they are specified explicitly (e.g., in the manifest file for Android apps) at install time. Furthermore, we presented the user with the inferences that could be made using the accessed sensors of each app, instead of merely listing sensors. To translate the sensor usage information to a list of possible inferences, we maintain an inference database generated from a survey of 60+ papers published in relevant conferences and journals over the past 3-5 years. New inferences can also be added to the database through our crowdsourcing web interface.

In ipShield, users can specify their privacy preferences in the form of a prioritized blacklist of private inferences and a prioritized whitelist of allowed inferences. The recommendation engine will then automatically generate lower-level binary privacy actions (e.g. suppression, allow) on individual sensors. ipShield also supports an advanced mode for manual configuration of fine-grained privacy rules for accessed sensors on a per app basis at runtime. Finally, ipShield provides the user with options to configure context-aware fine-grained privacy actions on different sensors on a per app basis at runtime. These actions range in complexity from simple suppression to setting constant values, adding noise of varying magnitude, and even playback of synthetic sensor data.

IV. Discussion and Conclusion

Each building block described in Section III enriches the sensing stack with a set of specific functionalities. By exploiting the processor heterogeneity of mobile SoC, context inference apps can be executed efficiently on IoT devices. Using the UniDev framework, a wide variety of sensors on IoT devices can be seamlessly incorporated into the stack for sensing and inference tasks. The FlowEngine provides rich user context and behavior inferences as a shared system service. With sensing pipelines customized by the app developers, the sensing stack generates and executes optimal sensing dataflow graph. Finally, ipShield acts as the privacy firewall in the sensing stack. All of the source codes are publically available online at https://github.com/nesl.

We are currently working on the design and implementation of an integrated sensing stack for the Internet of Things. The experience with this work also motivates several research questions about the IoT sensing stack, including leveraging low-power co-processors for real-time reinforcement of machine learning models; enabling app developers to specify their requirement about the accuracy, latency, and energy consumption of the inference; scheduling the inference execution across various IoT devices while leveraging processor heterogeneity; and adaptively migrating tasks between edge devices and the cloud.

V. Acknowledgment

This material is based in part upon work supported by U.S. ARL, U.K. Ministry of defense (MoD) under Agreement Number W911NF-06-3-0001, by the NSF under awards #0910706, #1029030, and #1213140, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via US Navy (USN) SPAWAR Systems Center Pacific (SSC-Pac), and by Qualcomm Inc. Any findings in this material are those of the author(s) and do not reflect the views of any of the above funding agencies. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Supriyo Chakraborty, Chenguang Shen, Kasturi Rangan Raghavan, Yasser Shoukry, Matt Millar, and Mani Srivastava. "ipShield: a framework for enforcing context-aware privacy." In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, pp. 143-156. USENIX Association, 2014.
[2] Chenguang Shen, Supriyo Chakraborty, Kasturi Rangan Raghavan, Haksoo Choi, and Mani B. Srivastava. 2013. Exploiting processor heterogeneity for energy efficient context inference on mobile phones. In Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower '13). ACM, New York, NY, USA.
[3] Jun Han, E. Owusu, L.T. Nguyen, A. Perrig, and J. Zhang. "ACComplice: Location inference using accelerometers on smartphones." In Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on, pp. 1–9, 2012.
[4] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. "Using mobile phones to determine transportation modes." ACM Transactions on Sensor Networks (TOSN), 6(2):13, 2010.
[5] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. "Tapprints: Your Finger Taps Have Fingerprints." In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, pp. 323–336, 2012.
[6] Md. Mahbubur Rahman, Amin Ahsan Ali, Kurt Plarre, Mustafa al'Absi, Emre Ertin, and Santosh Kumar. "mConverse: Inferring Conversation Episodes from Respiratory Measurements Collected in the Field." In Proceedings of the 2Nd Conference on Wireless Health, WH '11, pp. 10:1–10:10, 2011.
[7] Emre Ertin, Nathan Stohs, Santosh Kumar, Andrew Raij, Mustafa al'Absi, and Siddharth Shah. "AutoSense: unobtrusively wearable sensor suite for inferring the onset, causality, and consequences of stress in the field." In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys '11, pp. 274–287. ACM, 2011.
[8] Robert LiKamWa, Yunxin Liu, Nicholas D. Lane, and Lin Zhong. "MoodScope: building a mood sensor from smartphone usage patterns." In Proceeding of the 11th annual international conference on Mobile systems, applications, and services, MobiSys '13, pp. 389–402. ACM, 2013.
[9] David Chu, Nicholas D Lane, Ted Tsung-Te Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. "Balancing energy, latency and accuracy for mobile sensor data classification." In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, Sensys '11, pp. 54–67. ACM, 2011.
[10] Aman Kansal, Scott Saponas, AJ Brush, Kathryn S McKinley, Todd Mytkowicz, and Ryder Ziola. "The latency, accuracy, and battery (LAB) abstraction: programmer productivity and energy efficiency for continuous mobile context sensing." In Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications, OOPSLA '13, pp. 661–676. ACM, 2013.
[11] Snapdragon 800 MDP Mobile Development Platform. http://goo.gl/3UYXI
[12] TI Pandaboard. http://goo.gl/ujdiL.
[13] Apple M7 Co-Processor. http://en.wikipedia.org/wiki/Apple_M7.
[14] GOOGLE. "Activity Recognition API." http://goo.gl/mYJn84.
[15] 2013: The Internet of things, delivered via smartphone. http://goo.gl/GAkdCj
[16] A guide to the confusing Internet of Things standards world. http://goo.gl/hlMoCe
[17] Qualcomm Snapdragon. http://goo.gl/ZFTm0.
[18] TI OMAP. http://goo.gl/EIP6Zq.