# Extending Sensing Capabilities and Modalities of Mobile Devices

Rohit Chaudhri

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2014

Reading Committee:
Gaetano Borriello (Chair)
Richard Anderson
Shwetak Patel

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Extending Sensing Capabilities and Modalities of Mobile Devices

Rohit Chaudhri

Chair of Supervisory Committee
Professor Gaetano Borriello
Computer Science and Engineering

## Abstract

Mobile devices have a variety of built-in sensors (e.g., accelerometer, gyroscope, GPS, camera etc.) that allow them to be used in a wide range of context-aware mobile applications. Additionally, their built-in communication interfaces – like WiFi, Bluetooth and USB – let them communicate with external devices and sensors. Using built-in and external sensors, mobile devices help bridge the physical and digital worlds by providing precise readings on various natural phenomena (e.g., environmental and soil conditions).

Creating mobile sensing applications poses the following challenges for developers:

- Sensors often have only low-level communication interfaces; hence, an interfacing board is needed to connect them to mobile devices.
- Partitioning a sensing application between a mobile device running an OS like Android or iOS and an interfacing board that likely has an embedded OS at best, adds complexity to the system.
- In addition to implementing application logic, developers must overcome the quirks of different physical communication channels and process sensor-specific data.

My thesis addresses these challenges by: (1) extending the sensing capabilities and modalities of mobile devices, and (2) simplifying the development of mobile sensing applications. To this end, the thesis makes the following contributions:

- Hardware to enhance and simplify a mobile device's ability to connect to various types of sensors
- A software framework to simplify development of mobile sensing applications

- Implementation and evaluation of sensing systems to enhance workflows in low-resource settings

The research leading to my thesis explored different hardware options to interface low-level sensors to mobile devices of varying capabilities, ranging from low-tier, non-programmable phones to smartphones and tablets. I developed the Open Data Kit (ODK) Sensors framework for Android devices to simplify the development of mobile applications that interact with sensors. Using sensor interfacing boards and ODK Sensors as building blocks, I developed applications that have been deployed to address critical global health issues in developing countries. Further, the general-purpose technology building blocks are now being used in a variety of mobile sensing applications.

# Table of Contents


Chapter 1 Introduction ..... 1

Chapter 2 Sensor Interfacing Devices ..... 7
2.1 Sensing on Low-Tier Mobile Phones ..... 7
2.2 Sensing on Smartphones ..... 17
2.3 FoneAstra 3.0: Building Upon the Lessons Learned ..... 22
2.4 Related Work ..... 26
2.5 Conclusion ..... 30

Chapter 3 Open Data Kit Sensors Framework ..... 33
3.1 Motivation ..... 34
3.2 ODK and ODK Sensors ..... 35
3.3 Framework ..... 37
3.4 Applications ..... 47
3.5 Experiments and Evaluation ..... 49
3.6 Discussion and Future Work ..... 58
3.7 Related Work ..... 60
3.8 Conclusion ..... 62

Chapter 4 Vaccine Cold Chain Monitoring ..... 64
4.1 Motivation ..... 64
4.2 System Overview ..... 67
4.3 In-Lab Experiments ..... 71
4.4 Deployment in Albania ..... 75
4.5 Android-based Cold Chain Management ..... 85
4.6 Related Work ..... 90
4.7 Conclusion ..... 90

Chapter 5 Human Milk Banking ..... 92
5.1 Motivation ..... 92
5.2 The Human Milk Banking Process ..... 96
5.3 Mobile Device-based Milk Pasteurization Monitor ..... 97
5.4 In-Lab Experiments ..... 105
5.5 Deployment in South Africa ..... 111
5.6 Related Work ..... 121
5.7 Conclusion ..... 121

Chapter 6 Measuring Time-use ..... 124
6.1 Motivation ..... 125
6.2 System Overview ..... 127
6.3 Time-use Data Collection ..... 133
6.4 Deployment in Ethiopia ..... 135
6.5 Discussion ..... 141
6.6 Related Work ..... 142

6.7 Conclusion ..... 143

Chapter 7 Other Applications ..... 144
7.1 Emergency Tracheotomy Training Tool ..... 144
7.2 Cardiac Monitoring ..... 149
7.3 Prosthetic Monitoring for Gait Analysis ..... 151
7.4 mPneumonia ..... 154
7.5 Conclusion ..... 154

Chapter 8 Conclusion ..... 156
8.1 Insights on Creating Technologies for Developing Regions ..... 160
8.2 Future Work ..... 163
8.3 Final Remarks ..... 164

Appendix A: FoneAstra 3.0 Schematic ..... 165

Appendix B: Baseline Interview Questionnaire for Milk Bank Staff ..... 167

Appendix C: Post-trial Survey for Tracheotomy Project Evaluation ..... 173

Bibliography ..... 178

# List of Figures

Figure 1: The ODK system architecture ..... 3
Figure 2: The basic architecture of FoneAstra ..... 9
Figure 3: Physical realization of FoneAstra 1.0 ..... 10
Figure 4: The basic FoneAstra architecture enhanced with sensors ..... 10
Figure 5: The basic FoneAstra architecture enhanced with audio capabilities ..... 11
Figure 6: Physical realization of FoneAstra 2.0 ..... 12
Figure 7: Route approximation of an 8-mile round-trip in Bengaluru, India, generated using cell tower IDs recorded by FoneAstra ..... 13
Figure 8: Route approximation of a 45-mile trip from Bengaluru, Karnataka to Talai, a village in Tamil Nadu, India ..... 14
Figure 9: FoneAstra with a temperature sensor ..... 14
Figure 10: Motion sensor used to record time-use ..... 18
Figure 11: Researcher attaching time-use sensor to a water container ..... 19
Figure 12: Architecture of the USB Bridge ..... 20
Figure 13: Hardware setup on the Arduino USB Bridge ..... 21
Figure 14: The second version of the pasteurization monitoring system used in the human milk banking project ..... 21
Figure 15: The ATMega328p-based FoneAstra 3.0 board ..... 23
Figure 16: The third and final version of the milk pasteurization system installed at a human milk bank in South Africa ..... 24
Figure 17: End-user view of a smartphone using the ODK Sensors framework ..... 38
Figure 18: Architecture overview of the ODK Sensors system ..... 40
Figure 19: The three different ODK Senors framework implementations ..... 43
Figure 20: The V1 framework architecture incorporating four real-world applications ..... 49
Figure 21: Refrigerators and cold boxes used for vaccine storage and transportation ..... 65
Figure 22: Paper-based temperature reporting at a health clinic in Nicaragua ..... 67
Figure 23: FoneAstra with a waterproofed temperature sensor ..... 68
Figure 24: End-to-end architecture of the FoneAstra cold chain monitoring system ..... 69
Figure 25: Format of SMS messages sent from FoneAstra with an illustrative example ..... 70
Figure 26: A solar powered cold box enhanced with temperature monitoring ..... 71
Figure 27: Ground truth validation of FoneAstra's 1-wire temperature sensor ..... 72
Figure 28: Temperature curve of the cold box when its power was interrupted ..... 73
Figure 29: An experiment showing the possibility of detecting cold box lid open and close events based on its temperature profile ..... 74
Figure 30: FoneAstra deployed at the country's national store in Tirana, Albania ..... 76
Figure 31: FoneAstra deployed at a health center in Tirana ..... 77
Figure 32: The server's dashboard showing all the FoneAstra devices in the system ..... 78
Figure 33: Detailed view of the FoneAstra deployed at the DPH ..... 79
Figure 34: Freezing alarms reported by FoneAstra ..... 80
Figure 35: Temperature graph of a cold room at the IPH ..... 81
Figure 36: Temperature graph of a newly installed fridge ..... 82
Figure 37: Temperature profile of a poorly performing fridge ..... 83
Figure 38: A FoneAstra 3.0 board placed inside a waterproof Pelican case ..... 88

Figure 39: Temperature curve of the Styrofoam box while it was en route to Port Au Prince, Haiti ..... 89
Figure 40: A high-end commercial pasteurizer ..... 93
Figure 41: Low cost, low-tech flash heat pasteurization being used at a human milk bank 94
Figure 42: Typical temperature curve of milk during flash heat pasteurization ..... 95
Figure 43: The human milk-banking process ..... 96
Figure 44: Pasteurization monitor V1 ..... 98
Figure 45: An example of a table shown to survey participants regarding the most effective way to provide feedback from FoneAstra ..... 99
Figure 46: Screenshots of the Android application that guides the user through the pasteurization process. ..... 101
Figure 47: Archived pasteurization temperature curve accessible from the server. ..... 102
Figure 48: Pasteurization monitor V2 ..... 103
Figure 49: Pasteurization monitor V3 ..... 105
Figure 50: Experimental setup to understand the impact of the heat source on pasteurization ..... 106
Figure 51: Temperature curve obtained on a medium intensity flame ..... 107
Figure 52: Temperature curve obtained on a high intensity flame ..... 108
Figure 53: Temperature curve obtained on a low intensity flame ..... 108
Figure 54: A graph showing the responsiveness of the waterproof temperature probe used to monitor pasteurization ..... 109
Figure 55: A graph showing the temperature curve of water control and milk ..... 110
Figure 56: The 2nd step of the 2-step cooling process of pasteurization ..... 115
Figure 57: Pasteurization monitor V2 at the Neonate ICU of King Edward Hospital ..... 115
Figure 58: A screenshot of the Android Activity that lets technicians print pasteurization reports and labels for pasteurized milk jars ..... 116
Figure 59: A pasteurized milk jar with a label and a pasteurization report ..... 117
Figure 60: A user training session in the pediatrics lab at the UKZN. ..... 118
Figure 61: Water vessels commonly used in rural Ethiopia to collect water ..... 126
Figure 62: Pedometer data for a participant who reported collecting water for 90 minutes ..... 127
Figure 63: Motion sensor used to record time-use ..... 129
Figure 64: Time-use sensor's main-loop ..... 130
Figure 65: Battery life vs. time spent collecting water per day. ..... 131
Figure 66: Main Activity of the Android application that interacts with sensors. ..... 132
Figure 67: The Activity to retrieve trip data from the sensor ..... 132
Figure 68: Phase 1 of time-use data collection ..... 134
Figure 69: Phase 2 of time-use data collection ..... 134
Figure 70: Phase 3 of time-use data collection ..... 135
Figure 71: The Oromia region of Ethiopia where the system was trialed ..... 136
Figure 72: Researcher attaching time-use sensor to a water container ..... 137
Figure 73: Trips recorded by a sensor over a 3-day deployment at a household. ..... 139
Figure 74: A close-up view of the water collection round-trip recorded by the sensor ..... 139
Figure 75: Trips recorded by a sensor over a 3-day deployment at a household ..... 140
Figure 76: Disposable tracheal tube model ..... 145

Figure 77: Critical locations on the trachea, covered with conductive foils that act as cheap sensors ..... 145
Figure 78: Instruments used to perform cricothyrotomy ..... 146
Figure 79: The cricothyrotomy simulator mounted on a wooden board ..... 146
Figure 80: Screenshot from the Android application that guides users through a cricothyrotomy ..... 147
Figure 81: An early prototype of FoneAstra's single-lead EKG daughterboard ..... 150
Figure 82: Android screenshot of the EKG trace from a subject. ..... 151
Figure 83: A subject wearing a prosthesis that has been modified to include a force sensor ..... 152
Figure 84: The force profile when the subject was walking, constructed from the data received by the force sensor attached to the prosthesis ..... 153
Figure 85: A clinician using the mPneumonia Android app to get pulse oximetry data ..... 154

# List of Tables

Table 1: Comparison of capabilities of various phone architectures. ..... 16
Table 2: Performance evaluation of FROG. ..... 26
Table 3: Android constructs used in the three Sensors framework implementations. ..... 38
Table 4: Variations in the example sensing applications ..... 48
Table 5: Observed data throughput of four applications that receive sensor data from the framework ..... 50
Table 6: Throughput of the Bluetooth and USB Channels ..... 51
Table 7: Throughput results for a Nexus One when varying the packet size and inter-packet generation delay ..... 52
Table 8: Framework throughput of multiple sensors sending data every 64ms ..... 54
Table 9: Throughput of the V2 framework on different Android devices ..... 54
Table 10: Comparison of the lines of code needed to create a standalone sensing application vs. an application that uses the ODK Sensors framework. ..... 56
Table 11: Additional modules used in a Bluetooth-based sensing application written without using ODK Sensors. ..... 57
Table 12: Additional modules used in a USB-based sensing application written without using ODK Sensors.. ..... 57
Table 13: Data collected in Phase 1 of the time-use trial using an ODK form. ..... 133
Table 14: Data stored on a time-use sensor ..... 135
Table 15: Post-study questionnaire of the time-use trial ..... 138
Table 16: Tracheotomy tool evaluation survey items discussed in the section ..... 149

# Chapter 1

# Introduction

Mobile devices are rapidly becoming the platform of choice for deploying data collection and information services in the developing world. They have quickly leap-frogged desktop and laptop computers due to their mobility, increased independence from the power infrastructure, ability to be connected to the Internet via cellular networks, and relatively intuitive user interfaces that enable well-targeted applications for various domains. In effect, developing countries are using mobile devices for a variety of tasks that have traditionally been performed on larger machines. Further, cloud services are providing organizations with the ability to easily rent data storage space and scale hosting resources as needed, either locally or anywhere in the world.

The foundation of my work is based on two technology trends: (1) capable client devices with rich user interfaces, and (2) cloud-based scalable data collection, computing, and visualization services. Several years ago we began the Open Data Kit (ODK) [1] project at the University of Washington to leverage these trends. Through ODK, we sought to create an evolvable, modular toolkit for organizations with limited financial and technical resources to use for data collection and dissemination. We chose Android as our development platform because its flexible inter-process communication methods let us compose modular apps into larger systems rather than rewriting them ourselves, thus speeding development.

ODK's development was guided by a few simple principles, namely:

- *Modularity*: create composable components that could be easily mixed and matched and used separately, or together

- *Interoperability*: encourage the use of standard file formats and communication protocols to support easy customization and connection to other tools
- *Community*: foster the building of an open-source community that would continue to contribute experiences and code to expand and refine the software
- *Pragmatism*: deal with the realities of infrastructure and connectivity in the developing world and always support asynchronous operation and multiple modes of data transfer
- *Rich user interfaces*: focus on minimizing user training and supporting rich data types, such as GPS coordinates, barcodes, and photos
- *Follow technology trends*: use commodity consumer devices to leverage multiple suppliers, falling device costs, and a growing pool of software developers

ODK has quickly become one of the leading solutions for a wide variety of organizations, from small NGOs to large government ministries, and now boasts thousands of users in dozens of countries around the world. It is currently in use in domains such as global health (our original focus), environmental monitoring, and documenting human rights abuses. From our users' collective experience with ODK, we have identified many ways to improve the toolkit, expand its range of applications, and make it even more customizable. This vast amount of feedback, in conjunction with numerous feature requests submitted to our mailing list and website, led us to rethink the ODK system architecture.

Our refinement and expansion of ODK is based on four core design principles that are being incorporated into its tools:

1. When possible, UI elements should be designed using a more widely understood runtime language instead of a compile time language, thereby making it easier for individuals with limited programming experience to make customizations.
2. The basic data structures should be easily expressible in a single database row, and nested structures should be avoided when data is in display, transmission, or storage states.
3. Data should be stored in a database that can be shared across devices and can be easily extractable to a variety of common data formats.
4. New sensors, data input methods and data types should be easy to incorporate into the data collection pipeline by individuals with limited technical experience.

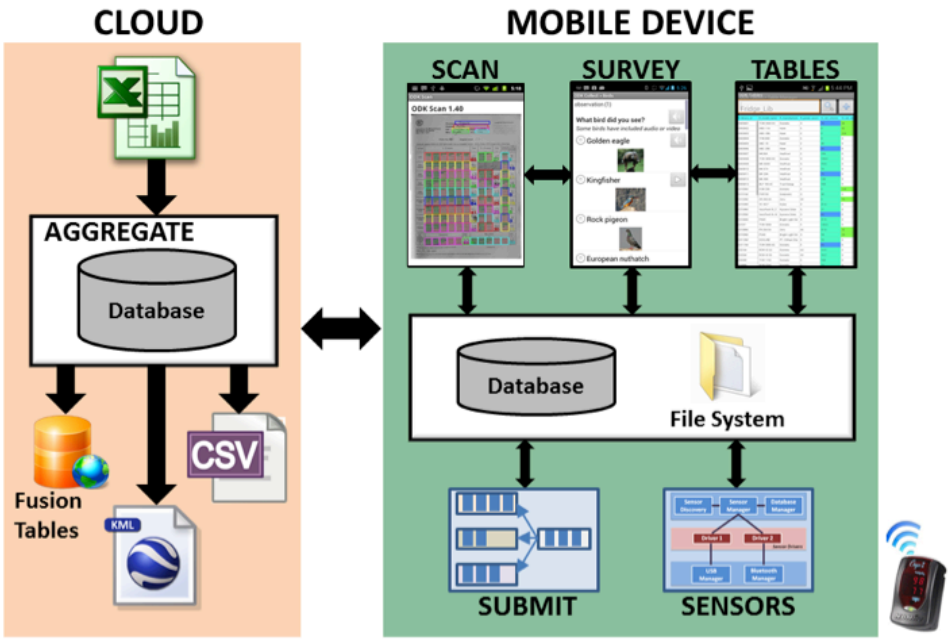



**Figure 1: The ODK system architecture, showing cloud services (left) and mobile client services (right). In the cloud, Aggregate provides services to synchronize data across devices and export data in common file formats. On the device, a common database/file system is shared between the tools and across clients. Scan, Survey, and Tables (above the database) are tools for gathering, processing and visualizing input data; Submit and Sensors (below the database) are tools that augment Android to create additional services.**

The ODK system architecture is premised on the belief that to be useful in developing regions, information services must be composable by non-programmers and deployable by financially and technically resource-constrained organizations that primarily rely on consumer services and devices. Therefore, the new ODK 2.0 toolkit (shown in Figure 1) provides a way to synchronize, store and manipulate data in *Tables* on a mobile device with a user interface that supports both the smaller smartphone screen and the larger tablet form-factors*; Tables* allows viewing and manipulating data in a simple row format. In addition, the new design makes customization easier by using widely understood standard presentation languages, such as HTML and JavaScript, to facilitate a more easily tailored user experience on a per *Survey* basis. Furthermore, ODK 2.0 makes it possible to attach external *Sensors* to mobile devices over both wired and wireless communication channels, thereby reducing the amount of manual data transcription from sensors into survey forms; it also facilitates the automatic conversion of information recorded on paper forms to a digital format by using the mobile device's camera to

*Scan* documents. Finally, *Submit* uploads data to the cloud using the most appropriate communication method available at any given time (e.g., 3G, SMS, WiFi).

This thesis describes how I extended the sensing capabilities of mobile devices through hardware and software. Originally, ODK developers assumed that people would enter data explicitly or, at most, gather data from built-in device sensors (such as GPS coordinates, barcodes, photos, audio, and video). The system design did not support the ability to interact with new sensors or process data captured from built-in sensors, like the camera. However, gathering information from external sensors is an often-requested feature; such requests range from the desire to enhance a health survey with data obtained from medical devices, to the ability to automatically incorporate GPS and compass data with captured photos. For instance, one user told us: *Our [use case] requires us to measure the height of trees. We currently use a clinometer for this and enter the data manually. It would be great if we could access the clinometer [from the device] and use it as part of our data collection process*.

ODK 2.0 reduces the amount of manual, and therefore error-prone, data transcription from sensors into surveys by making it possible to interface with external sensors. Sensors help bridge the physical and digital worlds by providing precise readings on various phenomena (e.g., one's location or pollutants in environment) that may be inaccurately described by humans. Additionally, difficulties such as training users to operate sensors and recognize bad data readings can be avoided.

Current generation mobile devices already incorporate several sensors (e.g., camera, GPS, accelerometer); this has lead to new, sensor-based applications (e.g., [2], [3]). With built-in communication interfaces like Bluetooth, WiFi and USB, however, mobile devices offer an attractive platform for building sensing applications that interact with external sensors as well. But building such applications can be complex because:

- Sensors often have only low-level communication interfaces; hence, an interfacing board is needed to connect them to mobile devices.
- Partitioning a sensing application between a mobile device running an OS like Android or iOS and an interfacing board that likely has an embedded OS at best, adds complexity to the system.

- In addition to implementing application logic, developers must handle the quirks of different physical communication channels and process sensor-specific data formats.

These technical barriers are exacerbated in many developing regions where technical human resources capable of building and supporting such systems are relatively scarce. To be practically scalable, the technology should work on existing mobile platforms without requiring significant modifications (e.g., rooting or jail-breaking the devices). Additionally, in these areas, such systems should be easy to deploy by organizations that might have only lightly trained personnel. The systems must be robust, with intuitive user interfaces that give clear indicators of failure (and how to rectify them) as deployments, done in real-world settings, often address critical issues specific to developing regions.

The research leading to my thesis explored different hardware options to interface low-level sensors to mobile devices of varying capabilities, ranging from low-tier, non-programmable phones to smartphones and tablets. I developed the ODK *Sensors* framework for Android devices to simplify the development of mobile applications that interact with sensors. Using sensor interfacing boards and ODK Sensors as building blocks, I developed applications that address critical global health issues and have been deployed in developing countries. The general-purpose technology building blocks are now being used for a variety of mobile sensing applications, more than I had initially envisioned.

My mission and motivation in writing this thesis, then, can be summed up as follows:

***Simplifying connections between mobile devices and sensors leads to better measurements that can improve global health systems.***

The contributions of my thesis are:

1. Hardware to enhance and simplify a mobile device's ability to connect to various types of sensors ([4]–[7])
2. A software framework to simplify development of mobile sensing applications ([5], [6], [8])
3. Implementation and evaluation of sensing systems to enhance workflows in low-resource settings ([9]–[13])

This thesis is organized as follows. Chapter 2 presents my exploration and subsequent technical evolution of sensor interfacing approaches. Chapter 3 discusses ODK Sensors, an Android framework that lets applications access both built-in and external sensors connected

over communication channels such as Bluetooth and USB. Chapters 4, 5, and 6 describe specific applications I built using the technology building blocks that have been trialed in developing countries (vaccine cold chain monitoring in Albania, human milk banking in South Africa, and time-use measurements in Ethiopia). Chapter 7 reviews other applications in various stages of development that also use these building blocks. Finally, Chapter 8 summarizes the insights I gained from creating technologies for developing regions and explores future directions for this work.

Chapter 2

# Sensor Interfacing Devices

We developed a set of sensor-interfacing devices to enable a variety of mobile sensing applications. Some add sensing capability to low-tier, non-programmable phones, while others extend the sensing capabilities of mobile devices (e.g., smartphones and tablets) by interconnecting sensors with low-level interfaces to the mobile. We discuss the hardware evolution and software elements of our sensor-interfacing devices in this chapter. We use the term "interfacing device" (or "interfacing board") interchangeably with "bridge" in this chapter as these devices act as a bridge between mobiles and sensors. This work provides evidence for contribution 1 of the thesis: **Hardware to enhance and simplify a mobile device's ability to connect to various types of sensors.**

Section 2.1 presents early work focused on adding sensing capabilities to low-tier mobile phones that are popular in low-income populations of developing countries. We then developed ways to extend the sensing capabilities of smartphones (and tablets), which are now becoming more common. This work is discussed in Section 2.2. Lessons learned from these efforts led to a major redesign of our sensor-interfacing platform, described in Section 2.3. We discuss related work in Section 2.4 and conclude in Section 2.5.

## 2.1 Sensing on Low-Tier Mobile Phones

Fieldwork done by colleagues in 2009 ([14]) indicated that low-tier phones (especially Nokia phones) were the most prevalent amongst people from low-income groups in both rural and urban areas of developing countries. While these phones (e.g., Nokia 1110, 1200, 1650, etc.) are cheap and affordable, they have minimal hardware and support only basic telephony. Most lack a programmable runtime environment as is available for Android, iOS, Windows Mobile, and Java ME (Micro Edition) on mid to high-tier phones. This restricts the services that can be delivered to the numerous users of low-tier phones.

Motivated by this observation our initial work focused on extending the capabilities of low-tier, non-programmable Nokia phones. Since software-only extensibility was not possible, our approach combined hardware and software to achieve the additional functionality. Our system, called FoneAstra, is a programmable accessory based on an ARM7 processor that connects to a phone through the phone's data port and communicates with it using a serial protocol. The serial interface exposes some cellular capabilities such as sending/receiving SMS, making/receiving phone calls, querying the cellular network for information, etc. We use the device for application-specific computation and leverage the phone for user input/output and cellular communications. This approach requires no customization at all on the mobile phone. FoneAstra costs only $15 USD in prototype quantities.

### 2.1.1 FoneAstra 1.0

The first version of FoneAstra was based on NXP's LPC2148 processor [15], an ARM7-based microcontroller that we configured to run at 60MHz. The ARM7 processor was chosen because of its low cost, support for low-power modes, and rich I/O interfaces (UART, SPI, $I^2C$, USB and GPIO). While cheaper processor options were available, we selected a more expensive, mid-tier processor since FoneAstra had to host an entire application: no processing could be offloaded to the connected (non-programmable) phone. We added a memory card interface to enable persistent storage of data. We also added two LEDs to provide simple visual feedback about the board's status. One of the two UARTs available on the LPC2148 was connected to the mobile phone's serial port, while the other was used for firmware upgrades and application debugging. FoneAstra can be powered with its own battery or share the phone's battery. In its active-mode (e.g. when sending/receiving SMS), FoneAstra draws about 50 milli-amps of current; in its low-power mode, it draws under 100 micro-amps. [16] provides details of the platform, including the schematics.

The serial interface exposed by low-tier mobile phones is the key to extending the phone's capabilities. Such phones typically implement an AT command interface ([17]) that is accessible over a serial link. Additionally, if AT commands are not implemented on the phone, vendors usually implement their own proprietary protocols. For instance, low-tier Nokia phones implement the FBUS (Fast-Bus) ([18]) serial protocol instead of the standard AT command-set. These serial interfaces expose a rich set of capabilities, including the ability to query phone information – such as the phone's IMEI identifier and phone number – send and receive SMS

messages, and initiate and answer phone calls programmatically over the serial link. This lets FoneAstra communicate with other users' phones as well as with services hosted in the cloud. Cellular information, such as the current cell tower-ID and signal strength, is also accessible over this interface; this information can be used to build location-aware services. Key presses on the phone generate DTMF tones that the device captures using a DTMF decoder. This enables the use of key presses for user interactions with FoneAstra. Since FoneAstra can also be battery powered, it can function independently when not connected to a phone. In this "disconnected" mode of operation, FoneAstra can be used to aggregate data in the field and, when it is later connected to a phone, use the newly available connection to upload the data to a remote server.

FoneAstra implements a subset of the FBUS protocol to communicate with low-tier Nokia phones. The supported phones include models released in 2009-2010, such as the 1209, 1661/2, 1650, etc., as well as older models (e.g., the 1110 and 1200).

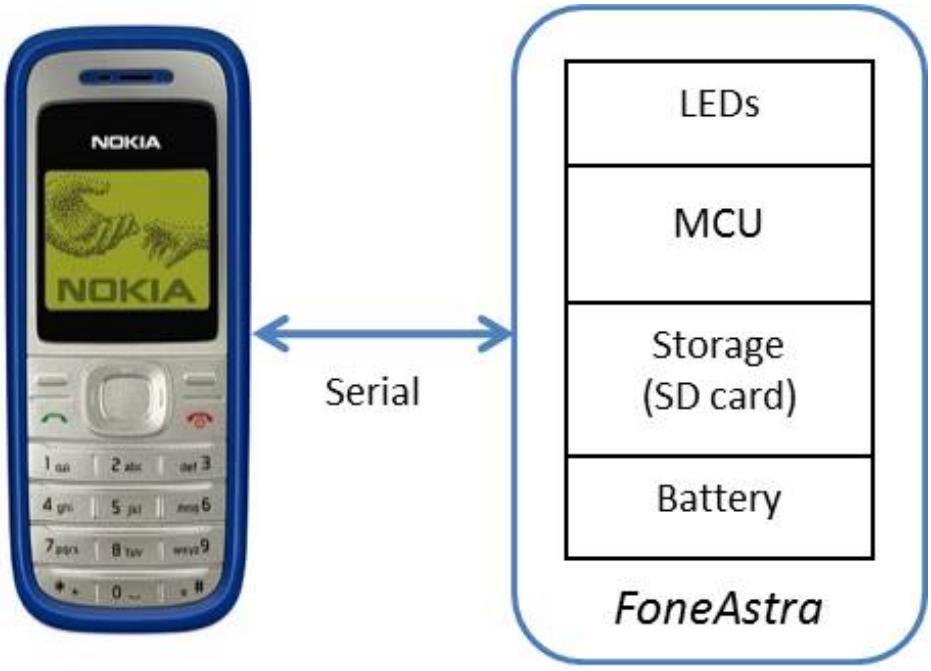



**Figure 2: The basic architecture of FoneAstra. FoneAstra is a battery-powered board based on the LPC2148 processor; it includes an SD memory card slot and 2 LEDs. It communicates with a low-tier Nokia phone over a wired connection using the FBUS serial protocol.**

Figure 2 shows the basic system architecture in which FoneAstra is connected to a mobile phone over the data port (this connector is called a "Pop Port" on low-tier Nokia phones). The phone's serial command interface is accessible over this link. Figure 3 shows the physical realization of the basic architecture.

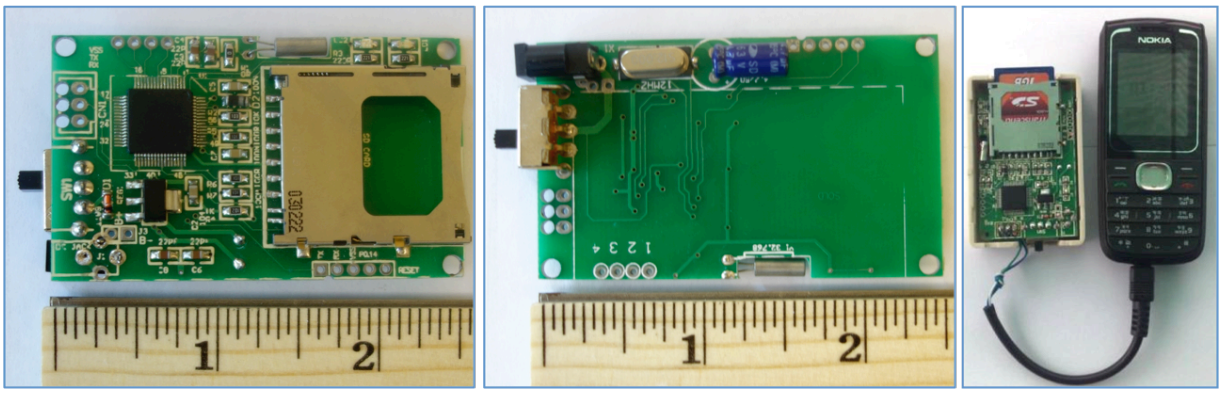


**Figure 3: Physical realization of FoneAstra 1.0. (Left) Front-side of the FoneAstra board, which has the LPC2148 processor, voltage regulator, 2 surface mounted LEDs, and SD memory card slot. (Center) Backside of the FoneAstra board, which has the power switch, DC jack and crystal oscillator. (Right) FoneAstra with an SD card, housed in a white enclosure, and connected to a Nokia 1650 phone over the serial interface.**

In Figure 4 we see the basic architecture enhanced with sensing capabilities. In this system, sensors are connected to the processor's peripheral interfaces. An implementation of this architecture was used for vaccine cold chain monitoring (cf. Section 2.1.3.2).

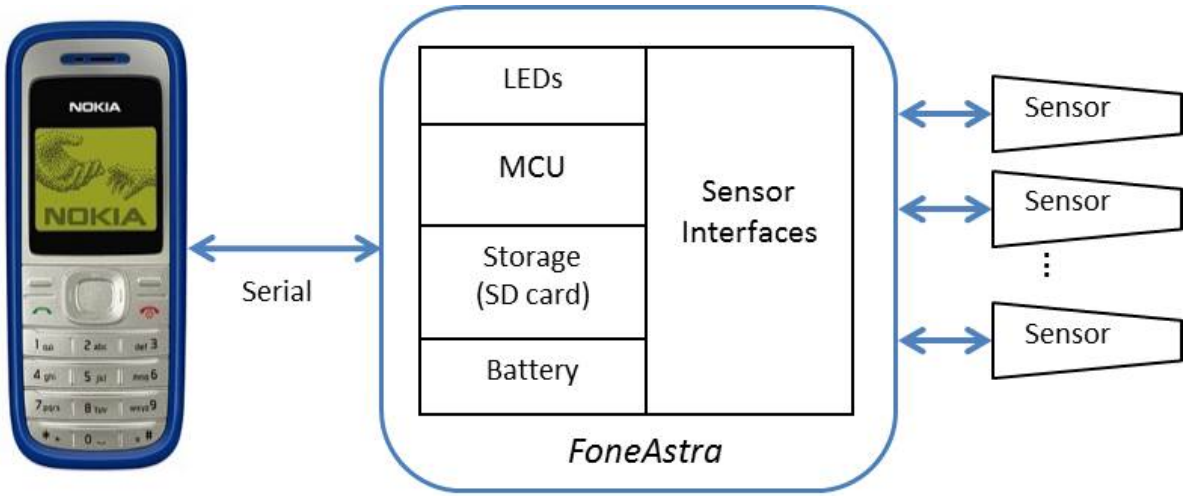



**Figure 4: The basic architecture of FoneAstra enhanced with sensors. The LPC2148 has several low-level digital and analog I/O pins that can be connected to sensors.**

Figure 5 shows the basic architecture enhanced with audio capabilities. This enables FoneAstra to accept and interpret numeric keypad input from the phone. When the user is on the phone's home screen, each numeric key press on the phone generates a unique DTMF tone. In addition to being transmitted over the cellular channel (when a voice call is active), these tones

are also sent over the phone's audio port. Hence, as shown in the figure, if the phone's audio port is connected to FoneAstra, it can capture and decode key presses from the phone. Additionally, FoneAstra can be connected to a headset enabling it to send audio prompts to the user. This capability would be especially useful for illiterate/semi-literate users. The configurations of Figures 2, 4, and 5 highlight different levels of capabilities possible with the same basic approach.

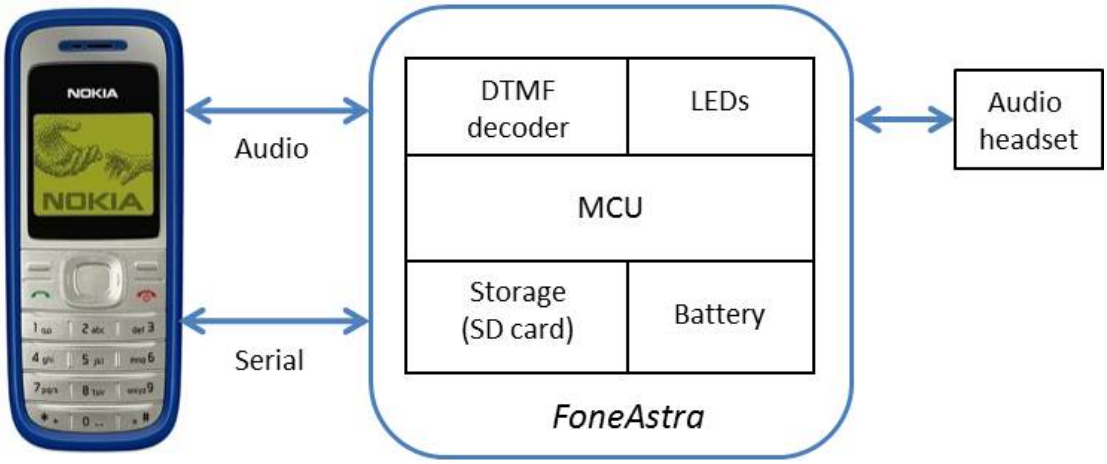



**Figure 5: The basic FoneAstra architecture enhanced with audio capabilities. FoneAstra has an audio jack that is connected to the headset port of the mobile phone. A DTMF decoder on the board receives and decodes DTMF tones generated due to key presses on the phone. A decoded DTMF tone identifies the key that was pressed on the phone and can be used to send input commands to the board. FoneAstra is also connected to an audio headset meant to be worn by the user; this allows the board to send audio prompts to the user.**

The software elements of FoneAstra were built on top of FreeRTOS [19], an open-source, general-purpose embedded real-time operating system. The system consists of a set of cooperative tasks that execute concurrently in the FreeRTOS runtime environment. For sensing applications, much of the application-specific logic was built into a task that periodically sampled the on-board sensors. Another task listened for and processed incoming SMS messages, while yet another controlled the on-board LEDs.

### 2.1.2 FoneAstra 2.0

In version 2.0 of FoneAstra we reworked its power subsystem to allow the Li-Ion battery to be recharged safely, independent of the phone. We added a dedicated port for the 1-wire sensors we needed for our applications. We also added additional feedback capabilities in this

version: a 2-line LCD, more prominent LEDs, and an audio buzzer. The LPC2148 can act as a USB device, so we exposed this capability via a mini-USB connector. Figure 6 shows the V2 implementation. We used FoneAstra 2.0 as the first prototype for the human milk banking application (cf. Section 2.1.3.3).

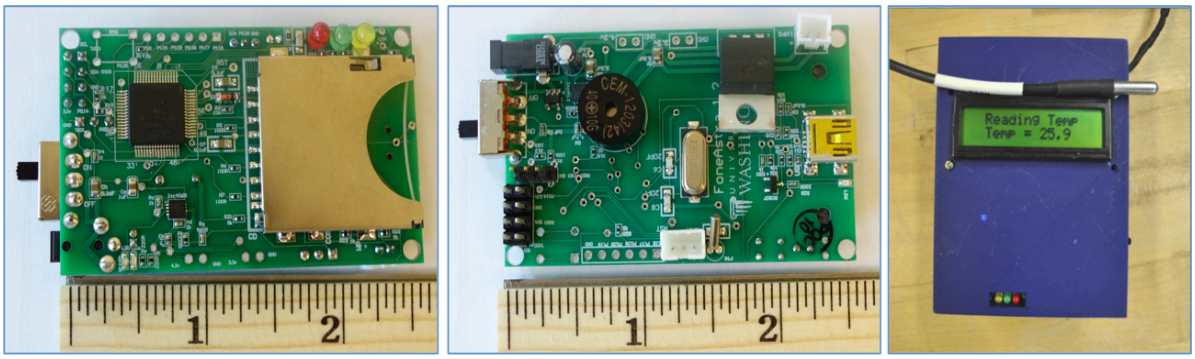



**Figure 6: FoneAstra 2.0. (Left) Front-side of the FoneAstra board has the LPC2148 processor, Li-Ion battery charging chip, 3 through-hole LEDs, and an SD memory card slot. (Center) Backside of the FoneAstra board has the power switch, DC jack, voltage regulator, audio buzzer, 1-wire and battery connectors, mini-USB port, and crystal oscillator. (Right) A Nokia phone and FoneAstra are inside the blue enclosure, while the LCD and LEDs are visible from outside the enclosure. A waterproof 1-wire temperature sensor is also connected to the FoneAstra board.**

## 2.1.3 Evaluations

We evaluated FoneAstra versions in the context of several different applications that are discussed below.

### *2.1.3.1 Location Tracking*

As mentioned earlier, FoneAstra can access the connected phone's cell tower-IDs over its serial interface. We used this information to create a location-tracking application for the Nokia 1650 connected to a FoneAstra 1.0 board. The 1650, a low-tier phone, lacks a GPS radio or a programmable runtime system to aid in collecting location data. In this application, FoneAstra was programmed to query the phone's current cell tower ID every two minutes and store this information on its memory card. The information was processed offline to obtain the location of cell towers from a Google web service. Valid requests sent to this web service contain a cell tower's ID and its Location Access Code; this information uniquely identifies the cell tower. Valid responses from the web service contain the latitude and longitude of the cell tower.

Figure 7 compares the locations of cell towers seen by the phone during an 8-mile round-trip in Bengaluru (Bangalore), India, to show the route of the same trip generated by Google

Maps. These figures show that the trip approximation obtained using cell tower IDs is very close to the trip-route generated by Google Maps (to within a few 100 meters).

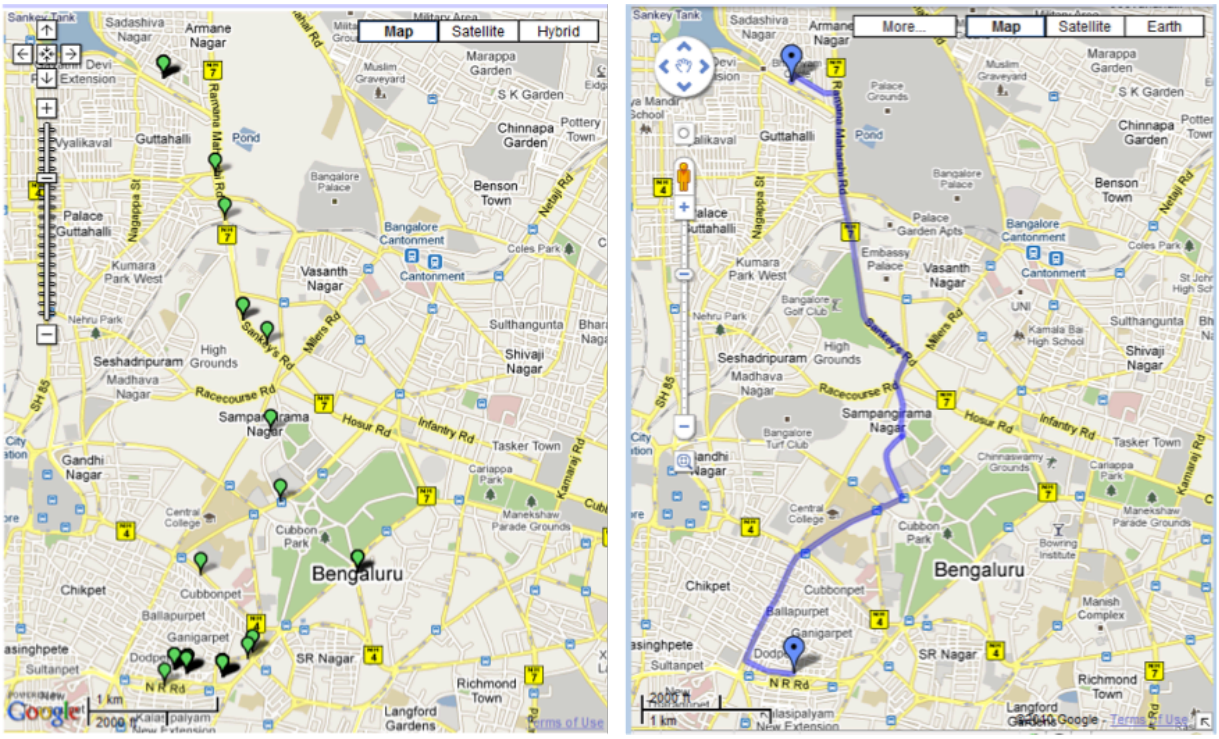



**Figure 7: (Left) Route approximation of an 8-mile round-trip in Bengaluru, India, generated using cell tower IDs recorded by FoneAstra during the trip. (Right) Trip generated by Google Maps for the 8-mile round-trip. These pictures show that the trip approximation obtained using cell tower IDs is very close to the trip-route generated by Google Maps (to within a few 100 meters).**

Figure 8 compares the location of cell towers, IDs for which were obtained during a 45-mile 1-way trip from Bengaluru to Talai, to the route generated by Google Maps for the same trip. The purpose of this trip, which covered some rural areas in the states of Karnataka and Tamil Nadu, was to determine if cell tower IDs could be used to effectively enable location tracking in remote, rural areas where cellular coverage is sparse. FoneAstra was powered off during parts of the journey that covered urban areas or a national highway. Hence, most of the cell tower markers are seen in the lower half of the figure, which covers the rural areas of this trip (the few markers in the Bengaluru area indicate where the trip started). Although the cell tower density is much lower in rural areas, the trip approximation is quite accurate (to within a few kilometers).

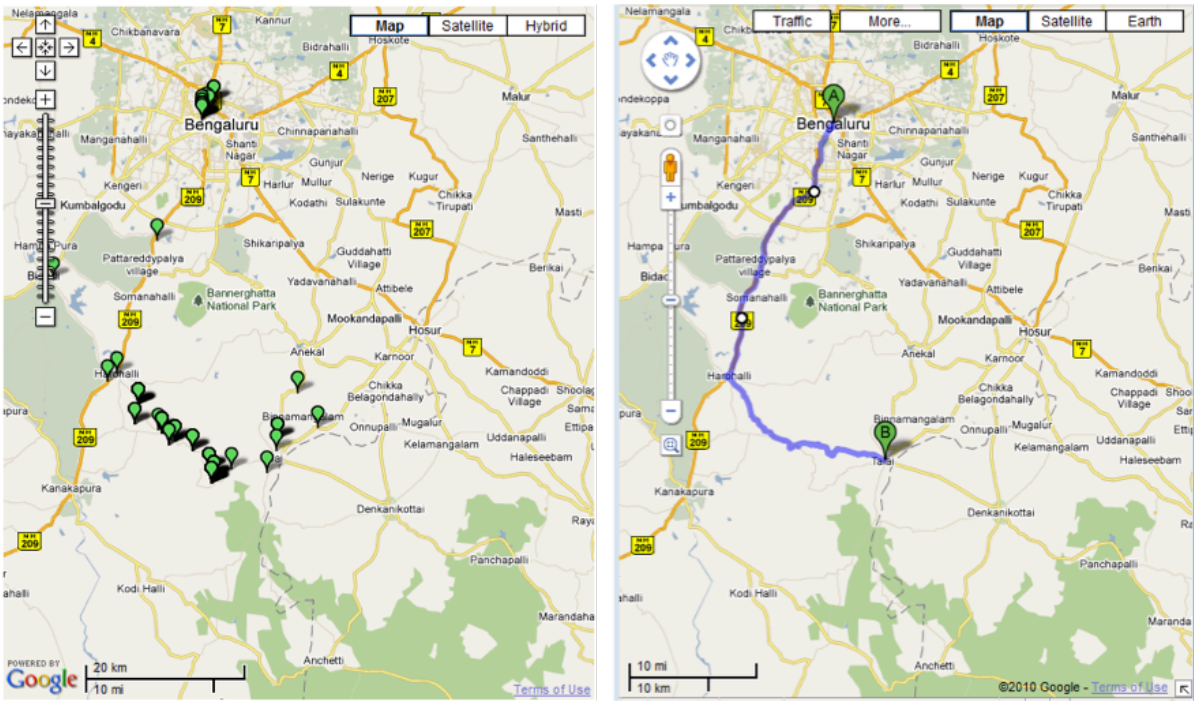



**Figure 8: (Left) Route approximation of a 45-mile trip from Bengaluru, Karnataka to Talai, a village in Tamil Nadu, India. (Right) Trip Generated by Google Maps for the same 45-mile trip. These pictures show that the trip approximation obtained using cell tower IDs recorded by FoneAstra is quite accurate (to within a few kilometers).**

### *2.1.3.2 Vaccine Cold Chain Monitoring*

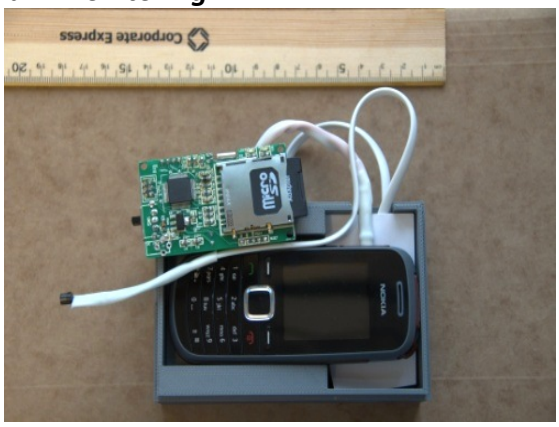



**Figure 9: FoneAstra with a temperature sensor (at the end of the white cable) and connected to a Nokia mobile phone. The ensemble, packaged in the gray box visible around the phone, has final dimensions of 12cm X 8cm X 5cm.**

In collaboration with PATH [20], a Seattle-based non-profit that works in the global health space, we trialed FoneAstra 1.0 to monitor parts of the vaccine cold chain in Albania and Nicaragua. For this application, multiple temperature sensors were connected to FoneAstra, and the device was programmed to periodically sample the sensors. Under normal conditions, the device periodically uploaded aggregated temperature data to a backend server (hosted in-

country) via SMS. Additionally, whenever temperatures deviated from a predefined normal range, the device sent immediate alert messages. The data was archived on the backend for visualization by cold chain supervisors. Figure 9 shows the monitoring system used for cold chain monitoring.

In the trial in Nicaragua, we monitored one fridge each at two health centers, and this trial lasted for a few weeks. The trial in Albania covered two storage rooms at the country's national level storage facility, two fridges at a district-level store, and one fridge each at three health centers. This trial lasted for six months. Both trials showed promising results, and the system was well received by the local cold chain authorities in the respective ministries of health. This work is discussed in more detail in Chapter 4.

#### *2.1.3.3 Human Milk Banking*

FoneAstra 2.0 was used as the first prototype to monitor breast milk pasteurization for the human milk-banking project (discussed in Chapter 5) and is shown in the preceding Figure 6. In this version, FoneAstra's temperature probe was placed in a glass jar containing breast milk, which was placed in a pan of water (the water-bath) that was heated to pasteurize the milk. The user turned FoneAstra on and started heating the water-bath. The temperature sensor was sampled every second, and the temperature was displayed on the LCD. FoneAstra's green LED blinked until the pre-configured high temperature threshold of 72°C was reached. At this point, FoneAstra started beeping and its red LED started blinking while the green LED was turned off. This told the user to stop heating and start cooling the milk. Temperature monitoring continued while the milk cooled down, and the procedure was considered complete when the temperature dropped down to 25°C. At this point the beeping stopped, the red LED turned off and the green LED turned on. FoneAstra also sent the entire pasteurization temperature curve to a server via several SMS messages.

We tested this device in Seattle to ensure its correct operation. A PATH collaborator (our partner in this project, as well) took the device to South Africa to get feedback from our in-country project partner there. They approved of the device and liked its simplicity and feedback mechanisms. However, our PATH colleague brought back new information about the human milk banking process, which gave us the intuition that a more capable mobile device would be better suited for this application to allow additional data to be recorded digitally. Therefore, we

stopped further work on the low-tier system, and started developing a smartphone-based solution, discussed in more detail in Chapter 5.

### 2.1.4 Summary

We initially envisioned two different kinds of applications for our system. First, FoneAstra could be used as an extension module for phones already carried by people. For instance, it could be used to track the location of users or be a platform for urban sensing via mobile users. However, the limited user interface and the wired connection between the mobile and FoneAstra made the system unwieldy for users, so we did not pursue such applications. Second, the system (phone + FoneAstra) could act as a programmable sensing and communications gateway for existing systems (e.g. asset tracking, cold chain monitoring, Wireless Sensors Networks, etc.). Such applications did not interact much with the user, beyond initial configuration, and they seemed better suited to our system. Hence, we pursued these applications further (e.g., cold chain monitoring). We also used this system to monitor the pasteurization of breast milk, an application that required some user interactions. However, we found our system's user interface to be limiting, so we switched to using Android phones for this application.

| Capability | Low- tier | Low-tier + FoneAstra | Mid-tier (Java ME) | High-tier |
|---|---|---|---|---|
| Send/recv SMS programmatically | No | Yes | Yes | Yes |
| Make/recv calls programmatically | No | Yes | Yes (make calls) | Yes |
| GPS | No | No | Optional | Optional |
| Location sensing by cell tower-ID | No | Yes | Optional | Yes |
| Log numeric key presses | No | Yes | Limited | Yes |
| Interfacing to external sensors | None | Extensible (UART, $I^2C$, SPI, ADC, GPIO) | Serial, BT, USB | BT, WiFi, USB |
| Extensible storage | No | Yes | Yes | Yes |
| Computational power | - | Fast | Slow | Fast |
| Daemon operation | No | Yes | Limited | Yes |
| Cost | $20 | $35 | >=$50 | >=$80 |

**Table 1: Comparison of capabilities of various phone architectures.**

Our system offered distinct advantages over other alternatives. Table 1 compares some relevant capabilities of our system with mid-tier Java ME and smartphones (e.g. Android, iPhone). Our platform has characteristics comparable to those of a smartphone at a lower price

point, while providing an extensible interface for connecting sensors. This makes it an attractive platform for building applications that require special-purpose sensing along with computing and communication capabilities. While Java ME phones offer a rich set of APIs, FoneAstra offers some capabilities that are not present in Java ME. First, Java ME offers limited or variable support for daemon operation, i.e., running programs in the background for continuous data collection (especially needed for continuous sensing applications). The Java ME standard has no provision for daemon operation, though some phone vendors have proprietary extensions to enable it. A second limitation of Java ME is performance; it may remain unsuitable for real-time operations, such as audio and speech processing. Campo et al. ([21]) compare the image processing performance of a Java ME midlet to that of a native Symbian application. They show that the native application performs significantly better than the midlet.

More recently, in 2011, the serial pop port was removed in some Nokia phones. Simultaneously, Android phones were getting much cheaper (e.g., Huawei IDEOS was available for $80 USD [22]). Recognizing that these trends would continue, we moved to Android phones to continue our mobile sensing work. Moving to a smartphone meant that we could simplify the interfacing board since the smartphone implements much of the application logic, and the board adds only the missing piece – i.e., the ability to interface with low-level sensors in a power-efficient manner.

## 2.2 Sensing on Smartphones

Smartphones already have some sensors built in (e.g., accelerometer, GPS) and can communicate with external sensors over standard interfaces like Bluetooth, WiFi and USB. However, most sensors available in the market have only low-level communication interfaces (such as $I^2C$ and SPI) and cannot be connected directly to a smartphone. An interfacing board is still needed to connect such sensors and obtain their data. This section presents some of the interfacing boards we built to enable sensor-based data collection on smartphones.

### 2.2.1 Sensor to Measure Time-use

Researchers at the University of Washington's Evans School of Public Affairs needed a way to measure the time people spend collecting water for daily use (generically referred to as a "time-use" study) in rural areas of developing countries. Such data helps policy makers and researchers understand the impact of providing improved sources of water. To aid with this data

collection, we built a Bluetooth-enabled motion-sensing device (Figure 10) that is attached to containers used for water collection. It records data in the on-board memory and the stored data is retrieved at a later time using a Bluetooth-enabled smartphone.

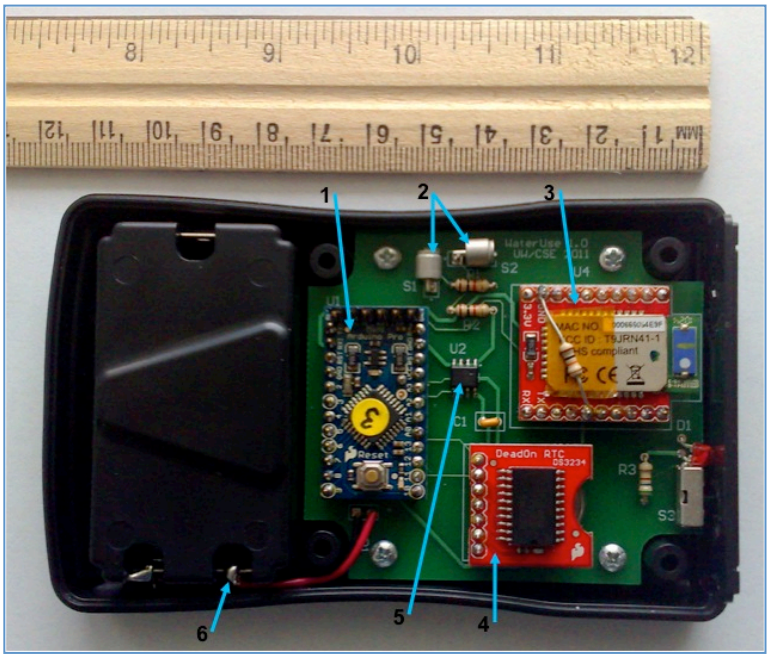



**Figure 10: Motion sensor used to record time-use. Its components are: (1) Arduino microcontroller, (2) mercury switches, (3) Bluetooth module, (4) RTC module, (5) EEPROM chip, (6) 2 AA Batteries (not visible on the left). The ensemble is packaged in a black box with final dimensions of 4.5" x 2.5" x 1".**

We chose the Atmel processor-based Arduino microcontroller for the sensing device because it is competitive in its energy efficiency compared to other microcontrollers and is easy to program. Even though low-power accelerometers are typically used for motion-tracking applications, we used small mercury tilt switches [23] because they required no power for operation. Mercury switches can be used as binary indicators of motion as they flip their state, which was sufficient for our purposes. Time-use trips are defined by a start-time and an end-time (4 bytes each). During testing, we verified that two mercury switches placed orthogonally to each other were sensitive enough to track motion for our application.

Our application required accurate timestamps to record time-use; since the Arduino does not have an internal real-time clock (RTC), we integrated an external RTC module on the board. An external 32KB EEPROM chip was also added to persistently store up to 2 months of time-use data. A Bluetooth module for communications with a Bluetooth-enabled mobile phone rounded out the board. Two AA Alkaline batteries provide 2700 milli-amp hours of power to the system

at 3 volts. These batteries were chosen for their ubiquity even in the developing world. The battery choice could change in the future (to rechargeable batteries), but it is straightforward simply to replace the batteries at the end of each data collection rather than taking the time to recharge them in the field.

**Figure 11: Researcher attaching time-use sensor to a water container. The lady is the primary water collector for the household. Photo credit: Yuta Masuda, UW, Seattle, USA.**

This system was trialed in Ethiopia in 2011. Figure 11 shows our motion sensor being attached to a water container. This work is discussed in more detail in Chapter 6. Even though this was an application-specific device, it was an important design point that helped guide the development of FoneAstra 3.0, described in Section 2.3.

### 2.2.2 USB Accessory for Android

Google announced the Android Accessory Development Kit (ADK) [24] in 2011, which enables programmatic access to compatible devices connected to an Android's USB port. We used an Android Open Accessory (AOA) Protocol-compatible Arduino to interface sensors to Android devices that implement the AOA protocol (i.e. have version 2.3.4 or higher of the

Android OS). Our implementation used an Arduino Mega2560 [25] as the sensor-interfacing USB Bridge. The Arduino served as the USB Host, enabling it to communicate with all Android devices that have a USB port and run a supported version of the Android OS.

We developed a lightweight software framework (depicted in Figure 12) for the Arduino board that implements this bridge. The Arduino Controller, an AOA wrapper class, handles USB communications with the connected Android device. The Sensor Manager maintains a mapping of sensor IDs and Sensor Drivers that communicate with sensors connected to the board over its low-level I/O interface. The Arduino framework is implemented in C++ in about 1000 lines of code, and the Sensor Drivers we have implemented to date are only about 50 lines of code each.

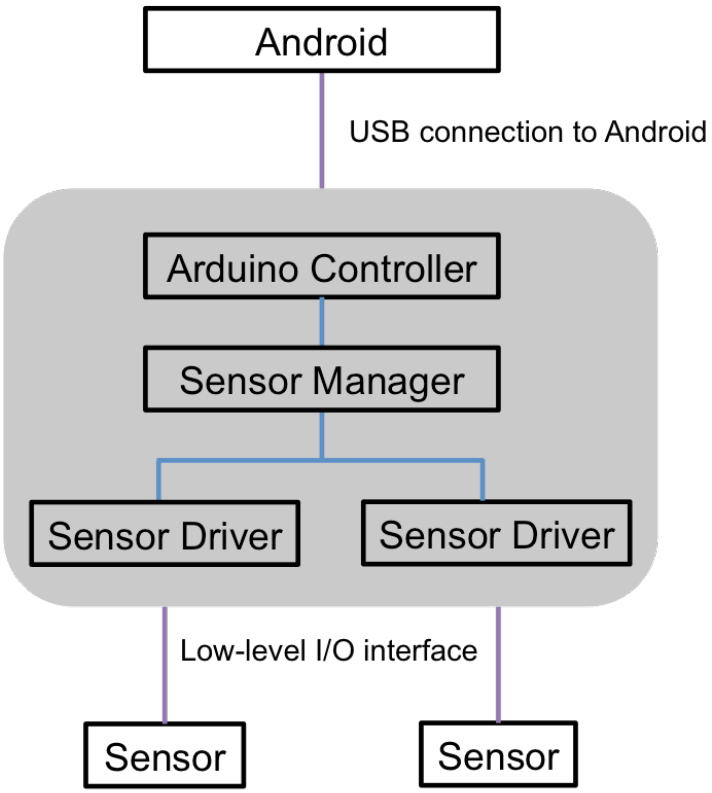



**Figure 12: Architecture of the USB Bridge. The gray box represents the Arduino Mega2560-based USB Bridge. Boxes inside the gray box represent software components. The Arduino connects to Android over USB and to sensors via its low-level I/O interface. The Arduino Controller facilitates communication between the Android and the Sensor Manager, which talks to each sensor through the sensor's driver.**

Figure 13 shows our Arduino-based USB Bridge that has a temperature and current sensor connected to it. This prototype was meant to be used for monitoring vaccine cold chains.

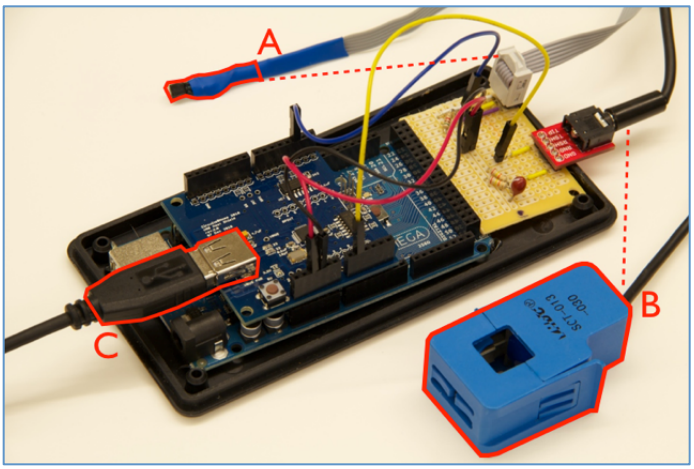



**Figure 13: Hardware setup on the Arduino USB Bridge showing: (A) temperature sensor, (B) current sensor, and (C) USB connection to Android.**

As mentioned in Section 2.1.3.3, the low-tier phone-based solution was not approprate for use in the human banking project due to its limited user interface. Therefore, we developed another version of the pasteurization monitoring device that was based on an Android and leveraged the USB Bridge to communicate with a temperature sensor. Figure 14 shows this pasteurization monitoring device, which has a temperature probe connected to an Android Nexus S phone via the USB bridge (inside the white enclosure below the phone). In 2012, we trialed this system at human milk banks in South Africa (cf. Chapter 5).

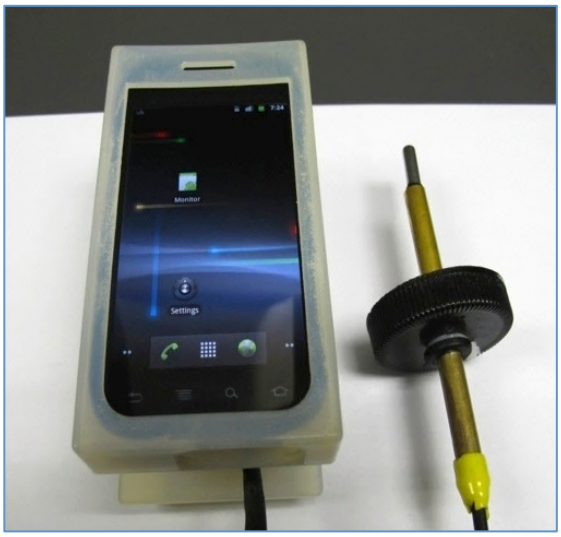


**Figure 14: The second version of the pasteurization monitoring system used in the human milk banking project. The monitoring app is implemented as an Android application that gets temperature readings from a probe connected to Android via a USB Bridge (inside the white enclosure and not visible in the picture). Cf. Chapter 5 for more details about the human milk banking project.**

Even though interfacing sensors to Android via a USB bridge was attractive, having this capability increased the cost and complexity of the interfacing board. Additionally, the AOA was available only on relatively newer and higher-end Android devices, which made the approach less suitable for the developing world. Due to these constraints, we moved away from this approach for interfacing low-level sensors to smartphones.

## 2.3 FoneAstra 3.0: Building Upon the Lessons Learned

Given the constraints of the USB accessory, in 2012 we decided to build another version of FoneAstra. Since our major focus was to interface with smartphones, we opted to use a simpler, Arduino-compatible processor (ATMega328p [26] running at 8MHz). This choice had significant advantages. First, since Arduino is very popular in the developer community, several software libraries were already available to interface with a variety of sensors. This dramatically reduces development time. Second, basing the board on an Arduino makes it attractive to use by the developer community since they are already familiar with the processor and the development environment. Third, the ATMega328p is a much simpler, lower-tier processor compared to the LPC2148 used in previous versions of FoneAstra. Its power consumption is also lower: it draws about 5 milli-amps of current in its active mode, and 2 micro-amps in its low-power mode, which are both orders of magnitude lower than the current drain of the LPC2148 (50 milli-amps and 100 micro-amps in the two modes, respectively).

We added a Bluetooth module to the board to enable it to interface with smartphones since Bluetooth seemed more appropriate than either USB or WiFi. We exposed the ATMega's UART interface on a connection header, allowing phones to interface over a serial interface, as well. This lets the board communicate with low-tier phones, much like the earlier versions of FoneAstra. Additionally, smartphones that have a USB-serial driver (e.g., Androids with USB Host capability) can also communicate with FoneAstra over this UART interface. Thus, FoneAstra acts as a USB Bridge as well as a Bluetooth Bridge for mobile devices.

FoneAstra provides connection headers for commonly used low-level communication protocols, such as $I^2C$, SPI and 1-wire, as well as general-purpose digital and analog I/O. Being battery-powered, it can operate autonomously, enabling it to support applications such as time-use studies, where a mobile device connects intermittently to a sensor board to retrieve sensed data buffered over a period of time. For applications such as human milk banking and vaccine

monitoring, in which a mobile phone is co-located with the board, FoneAstra has a second USB port that is used to power the co-located phone. When connected to a mobile device capable of being a USB Host, the board can be powered by the mobile device itself and does not require a battery. Additionally, FoneAstra has an on-board SD memory socket, a real-time clock (for autonomous data collection), a few LEDs, and an audio buzzer for feedback to users. Figure 15 shows the front and backside of this version of the board, and its schematics are available in Appendix A.

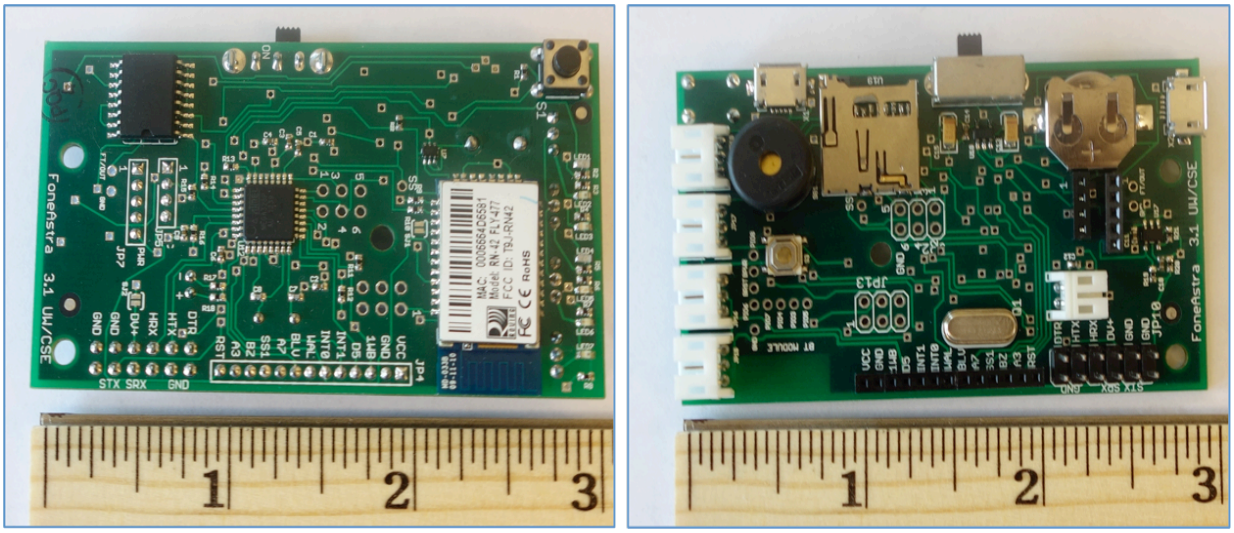



**Figure 15: The ATMega328p-based FoneAstra 3.0. (Left) Front-side of the board has the processor, Bluetooth module, RTC chip, and a tactile switch. (Right) Backside of the board has 4 1-wire sensor ports, $I^2C$, SPI, UART and other connection headers, battery connector, audio buzzer, micro SD card slot, 2 micro USB ports for charging, and crystal oscillator.**

FoneAstra 3.0 is currently being used in South Africa for the human milk banking application (cf. Chapter 5). Figure 16 shows the configuration from a human milk bank in South Africa. Due to the flexibility and generalization achieved by version 3.0, it is being used for a variety of mobile sensing applications (in addition to cold chain monitoring and milk banking) that are discussed in Chapter 7. The board was also used by the UW Deparment of Electrical Engineering in a graduate-level course on mobile signal processing. In this course, students built mobile-sensing applications on Windows Mobile phones that interfaced with external sensors via FoneAstra.


**Figure 16: The third and final version of the milk pasteurization system installed at a human milk bank in South Africa. It uses the FoneAstra 3.0 board to interface the temperature sensor to Android over Bluetooth. FoneAstra is inside a black enclosure that is mounted on the wall, and the Android phone is mounted on the enclosure. The inset shows a close-up view of the ensemble. Cf. Chapter 5 for more details about the human milk banking project. Photo credit: Paarl Hospital, Cape Town, South Africa.**

We plan to enhance FoneAstra in the near future. The Bluetooth 2.1 module [27] we currently use on the board consumes a lot of power. Since Bluetooth Low Energy (BLE) radios are becoming increasingly popular (and cheaper due to high volume production) in mobile/embedded devices, we will replace FoneAstra's Bluetooth module with a BLE module. We also plan to integrate a Near Field Communications (NFC) radio onto FoneAstra to provide a low-energy, low-bandwidth communication mechanism; as a result, NFC-enabled mobile devices will be able to communicate with the board over NFC (in addition to Bluetooth). For applications like time-use measurements (cf. Chapter 6), in which Bluetooth is not used very often, we envision using NFC to wake up the Bluetooth module whenever it is needed (e.g., to retrieve sensor data over Bluetooth). For instance, the Bluetooth radio could be turned on only when a mobile device scans FoneAstra's NFC module. Finally, we intend to make a daughterboard for FoneAstra that has an e-ink display. Since e-ink displays do not consume power in a steady state (i.e., when the display is not being refreshed), they are especially

attractive for providing visual feedback on battery-powered embedded devices, given, of course, that the display's refresh rate is not too high.

### 2.3.1 FROG: Simplifying Mobile Sensing

Currently, it is difficult for end-users to start using a mobile sensing application for their needs. The firmware running on the sensing hardware is either pre-programmed by the manufacturer, which makes after-sales firmware updates difficult, or technology-savvy end-users program sensing hardware themselves using computer-based software. The latter poses a barrier for end-users who lack technical skills.

We addressed this problem by developing an Android application called FROG (FoneastRa prOGgrammer) that lets mobile phone users update FoneAstra's firmware from Android over a Bluetooth connection. FROG implements the STK500 [28] protocol for programming Atmel processors and programs FoneAstra over a Bluetooth connection. We also modified FoneAstra's bootloader: upon reset, if the bootloader detects an active Bluetooth connection, it waits a few seconds longer to receive programming commands over the connection before timing out and jumping to the default application program. FoneAstra can be reset either manually via a power switch, or programatically via a command sent to the application over Blueooth.

Currently, FROG lets the end-user select a firmware file from the device's local memory and remotely reset FoneAstra to start reprogramming. We are also developing a mechanism to package FoneAstra's firmware file(s) in an Android application; the application can share those files with FROG to install onto the board. We envision that this capability will help create an ecosystem for mobile sensing applications that leverage interfacing devices like FoneAstra. Developers could create device software that leverages sensors, package the firmware file in an Android APK file, and publish it to an appstore. End-users would buy their interfacing device and the sensors they need for their application, download the appropriate firmware from the appstore, and use FROG to automatically install the firmware on their device. They could also update their firmware anytime developers publish new firmware for an application at the appstore. This model of firmware distribution simplifies customization for device manufacturers as well, as they can ship products with basic software and leave other software customizations up to end-users and developers.

FROG has a total of 1350 lines of code implemented in five Java files. We evaluated FROG by measuring the time it took to program FoneAstra with five firmware files of different sizes (1KB, 12KB, 40KB, 60KB, and 70KB). We compared its performance with that of AVRDUDE [29], an open-source computer-based software used to program Atmel microcontrollers. AVRDUDE was used on a Macbook Pro to program FoneAstra with the same files over a wired serial and a Bluetooth connection, while FROG was used on a Galaxy Nexus smartphone. Each file was programmed five times using each tool (i.e., FROG, AVRDUDE over wired serial, and AVRDUDE over Bluetooth), and the timings from the five runs for a file were averaged.

| | FROG | AVRDUDE (Wired) | AVRDUDE (Bluetooth) |
|---|---|---|---|
| **1KB** | 1.3 | 0.2 | 2 |
| **12KB** | 7.3 | 2.2 | 26.1 |
| **40KB** | 21.4 | 7 | 82.3 |
| **60KB** | 31.4 | 10.3 | 122 |
| **70KB** | 36.5 | 12.1 | 143 |

**Table 2: Performance evaluation of FROG. Five firmware files of different sizes were used to program FoneAstra using FROG over Bluetooth, AVRDUDE (running on a Macbook Pro) over a wired serial connection, and AVRDUDE over Bluetooth respectively. The firmware files are listed by their size (in kilobytes) in the leftmost column. The numbers in the other cells represent the average time (in seconds) it took to program a file using a particular tool. For all but the file that is 1 kilobyte in size, FROG was about three times slower than AVRDUDE over a wired serial connection, but it was about four times faster than AVRDUDE over a Bluetooth connection.**

Table 2 summarizes the results of these experiments. For all but the file that is 1 kilobyte in size, FROG was about three times slower than AVRDUDE over a wired serial connection, but it was about four times faster than AVRDUDE over a Bluetooth connection. We had expected that FROG would be slower compared to AVRDUDE over a wired connection, but the difference in performance between FROG and AVRDUDE over Bluetooth was surprising. Based on these results, we concluded that FROG's performance was acceptable since it addresses a critical barrier that will improve the distribution and maintenance of firmware for boards like FoneAstra.

## 2.4 Related Work

Work related to sensor interfacing devices for mobiles can be categorized into two broad areas: (1) extending the sensing capabilities of mobile devices, and (2) energy-efficient

continuous sensing on mobiles. This section discusses our work in the context of others' work in these two areas.

### 2.4.1 Extending Sensing Capabilities of Mobile Devices

Current generation mobile devices already have a variety of built-in sensors. However, most sensors available in the market have only low-level communication interfaces; hence, an interfacing board is needed to connect such sensors to mobile devices.

#### *2.4.1.1 Hijack*

Hijack [30] exploits the universal audio headset port on mobile devices to power and communicate with an external board that hosts sensors. This system generates an audio wave on one of the earphone channels to drive an energy harvesting circuit that powers a microcontroller (an MSP430) connected to the mobile device over the headset port. The second earphone channel is used to send data to the microcontroller, while the microphone channel is used to receive data from the microcontroller.

The Hijack power harvester delivers 7.4 mW of power, which is sufficient to drive the MSP430 and host low-power sensors. The system achieves a 1 Kbyte/sec data rate using duplex UART for communication. While the power generated is sufficient to drive low-power sensors connected to the MSP430, the system requires an audio signal to drive the power line, which raises the energy cost per sensor sample collected on the microcontroller and drains the phone's battery very quickly. An additional 200mW of power is required to play the audio signal that drives the MSP430. However, the system interfaces low-power sensors to mobile devices at low cost – the interfacing electronics are only about $2.34 at 10K volume. This makes it ideal for non-continuous sensing applications (i.e., where Hijack is plugged in to sense for a short period of time and then disconnected) that do not require extremely high data rates.

Hijack must be connected to the headset port in order to provide sensor interfacing; FoneAstra, on the other hand, enables a variety of sensing applications – long-running continuous or non-continuous apps, with or without a mobile device always co-located and connected, so it is more general-purpose. Powering FoneAstra from a USB Host-capable Android device is similar to the sensing system enabled by Hijack. While our system will be more energy efficient, and will achieve higher data rates, it will not be as ubiquitously useful as Hijack because USB Host (or USB On the Go) is not as commonly available on Androids yet.

#### 2.4.1.2 AudioDAQ

AudioDAQ [31] is more recent work from the group that built Hijack. Like Hijack, it also exploits the audio headset port on mobile devices to interface with external sensors. It enables analog sensors to be connected to mobile devices, and, compared to Hijack, is simpler, lower cost, and more energy efficient. It is also generalizable to a variety of mobile phone platforms (e.g., iOS, Android and even feature phones). Using the built-in voice memo application and decoding the sensed data in the cloud, AudioDAQ can be used for data acquisition from analog sensors without requiring hardware or software modifications to the mobile device.

AudioDAQ does not require a power harvesting circuit and targets a class of mobile-sensing applications that have a lower power budget than those targeted by Hijack. The system uses the microphone bias voltage to drive sensors and interfacing circuitry, and modulated sensor data is sent to the mobile device over the same microphone line. Multiple sensing channels can be supported on AudioDAQ; however, the power budget and circuit complexity increases with each additional channel. Signal reconstruction complexity also increases with each channel, especially because there is some coupling between the sensor channels. While AudioDAQ is more energy efficient than Hijack, the energy cost per sensor sample is still high, making it unsuitable for long-running, continuous sensing applications.

AudioDAQ has tradeoffs similar to Hijack when compared to FoneAstra.

#### 2.4.1.3 IOIO

IOIO [32] is a PIC-based development board designed to work with Android devices over a USB connection (or over Bluetooth via a USB dongle [33]). It abstracts the communications between sensors connected to the board and software running on the Android, enabling applications to directly control sensors attached to the IOIO board. It differs from the interfacing boards we developed because IOIO provides an abstraction to Android applications at the level of I/O pins on the IOIO board. By providing direct access to I/O pins at the Android-level, IOIO shields Android application developers from programming the PIC microcontroller. However, the energy cost of accessing sensors is fairly high in this system because sensor-sampling happens on the Android, which incurs a higher energy overhead compared to sampling sensors on a low-power microcontroller, buffering the sensor readings, and then periodically sending up the buffered readings to the mobile. Our approach provides low-power sensing by decoupling the interfacing board from the mobile application. An independently operating

interfacing board enables sensing to occur at much lower power, allowing the mobile device to remain in a sleep state longer; this is especially important for long-running continuous sensing applications like cold chain monitoring.

### 2.4.2 Energy Efficiency for Continuous Sensing on Mobiles

Motivated by the emergence of continuous sensing applications on mobile devices in recent years (e.g., [34], [35]), researchers have studied such systems to understand the implications for the devices' energy budget. They have proposed architectures to improve the energy efficiency of applications that interact with built-in sensors. Our work complements this area of work since we primarily focus on extending the sensing capabilities of mobile devices by providing ways to connect sensors that are not already built into the mobile. Because our interfacing boards are battery-operated, energy efficiency is definitely a consideration, but it is secondary. We can lower the overall energy budget because low-level tasks (like sensor sampling and data buffering) happen on the interfacing board, which uses significantly less power in its active state; doing these tasks on the mobile device consumes more energy in its active state. Thus, the mobile device can be in a low power state longer.

#### *2.4.2.1 Mobile Sensing Platform*

The Mobile Sensing Platform (MSP) [36] is a wearable device designed for activity recognition in order to support context-aware ubiquitous computing applications. The MSP was amongst the early platforms that enabled software-based activity recognition, before sensors started becoming available on mobile devices, making it possible to do initial research on the uses of sensors built into such devices. The platform is based on two boards: (1) a lighter weight, ATMega128 sensor board that hosts a variety of sensors (e.g., accelerometer, barometer, compass etc.), along with a memory card and Bluetooth radio, and (2) a richer XScale PXA271-based board that processes sensor data and executes activity inferencing algorithms.

The MSP has been used with mobile phones for several context-aware applications (e.g., [37], [38]). It is amongst the early works that paved the way for current trends in mobile device-based sensing and more recent trends in integrating low-power sensor-hubs into mobiles to enable energy-efficient continuous sensing mobile applications. Our work with sensor interfacing boards complelements the MSP work because we do not integrate sensors directly onto the boards; instead, our focus is on plugging in external, low-level sensors that are not already built into mobile devices.

#### *2.4.2.2 LittleRock*

Priyantha et al. [39] identify the causes of high power consumption on mobile devices used for continuous sensing applications. In current mobile device architectures, built-in sensors (e.g., accelerometer, GPS) are connected to the main, powerful application processor (AP) of the device that runs a standard operating system like Windows Mobile or Android. The energy cost to collect a sample from a sensor is high as the device needs to be in its active, high-power state when it is sampling sensors. Additionally, compared to simpler, low-power microcontrollers (LP), standard OSes take longer to transition between the high-power active and low-power sleep states (close to 1 second). As a result, even at low sampling rates the OS can never transition to its low-power state when a continuous sensing application is running.

They propose an alternate architecture in which built-in sensors are connected to an LP that is connected to the AP over a local communication bus. The LP has significantly lower power consumption in its active and low power modes, and it transitions between these modes very quickly. This architecture is realized using an MSP430-based platform called LittleRock. Continuous sensing applications running on the AP can configure sensors on the LP, allowing the LP to autonomously sample sensors and buffer the readings, while the AP transitions to its sleep state. The LP periodically interrupts the AP to wake it up, allowing the AP to retrieve and process sensors readings. Using a pedometer application as an example, the authors show that the LittleRock architecture is three orders of magnitude more energy efficient than a standard mobile device.

In more recent work [40], the LittleRock group proposed a layered set of APIs for the LP that are used by multiple apps on the AP, along with guidelines for when tasks executing on the LP should be evicted and moved to execute on the AP. While we do not have a concept of task eviction, our interfacing boards provide some of the lower-layer functionality mentioned in [40], including sampling and buffering of sensor data.

## 2.5 Conclusion

This chapter presented our work on sensor interfacing boards that extend the sensing capabilities of mobile devices. These boards are based on low-cost, low-power microcontrollers that have several digital and analog I/O interfaces for connecting to low-level sensors. The interfacing board acts as a bridge between such sensors and a mobile device. To date, we have

developed interfacing boards that connect to mobiles over wired (serial and USB) and Bluetooth connections.

We started our work in this area in 2009 by developing an interfacing board called FoneAstra, which was meant to extend the computing capabilities of low-tier, non-programmable mobile phones that were commonly used amongst low-income populations in developing countries at that time. The first version of FoneAstra used the ARM7-based LPC2148 processor to host application-specific sensors and software, and it connected to a low-tier phone over a wired serial interface to leverage its cellular capabilities (e.g., sending/receiving SMS, making/receiving phone calls and querying the phone's cell tower information). The complete package made for a low-cost, networked sensing platform. We added a temperature sensor onto the board and used the system to monitor parts of the vaccine cold chain in Albania and Nicaragua (cf. Chapter 4).

Driven by application requirements, we built another version of FoneAstra (v2.0) with enhanced feedback capabilities. Specifically, a 2-line LCD display, an audio buzzer, and more prominent LEDs were added in this version. Additionally, we added circuitry to allow the board's Li-Ion battery to be charged safely. This version also exposed the LPC2148's USB device capabilities via a mini-USB port. The first version of the human milk pasteurization monitor was based on FoneAstra 2.0 (cf. Chapter 5).

Later, in 2011, the falling prices of Android devices made them more appropriate for our work in developing countries. We also needed a richer user interface on the devices for creating sensing applications that interacted with users. Therefore, we switched to using Android devices for our mobile sensing work. However, we still needed a way to connect low-level sensors to Android, so we continued iterations on our interfacing board platform. First, we developed a USB-based board that communicated with the Android over a USB connection, using Android's Open Accessory protocol. An Android coupled with this board (that also had a temperature sensor) was used for the second version of the human milk pasteurization monitor that was trialed in South Africa in 2012 (cf. Chapter 5). We also developed a Bluetooth-enabled motion sensing board that was used in conjunction with Androids, which connected periodically to the board over Bluetooth to retrieve historical sensor data. Researchers working in the water sector used this system to measure the time spent by people collecting water for daily use in rural Ethiopia.

Experience gained from building these four boards led to the development of the third and final version of FoneAstra. This board combines the capabilities of the previous boards into one package and supports different classes of mobile sensing applications (cf. Section 3.4). It communicates with a mobile device over Bluetooth or USB and can be powered by a dedicated battery or via USB by devices that have USB Host capability. It provides dedicated ports for connecting 1-wire, $I^2C$, SPI, and UART devices, and all the processor pins are accessible via connection headers for interfacing devices that might require some other interface (e.g., an analog interface). FoneAstra is suitable for mobile sensing applications that process sensor data in real-time (e.g., vaccine cold chain monitoring and human milk banking), or applications that periodically retrieve historical sensor data (e.g., time-use measurements). In addition to vaccine cold chain monitoring (Chapter 4) and human milk banking (Chapter 5), FoneAstra is being used in several other applications discussed in Chapter 7. To simplify the deployment of mobile sensing applications, we also developed an Android application called FROG (FoneastRa prOGgrammer) that reprograms FoneAstra over a Bluetooth connection. This lets application developers include the board's firmware in an Android application published in app marketplaces like Google Play. We believe that FROG will simplify the distribution and maintenance of firmware for boards like FoneAstra and will be beneficial for sensor manufacturers, application developers and end-users alike.

The next chapter presents Open Data Kit (ODK) Sensors, an Android framework we created to simplify the development of mobile sensing applications. Interfacing boards described in this chapter work in conjunction with ODK Sensors to enable a wide range of applications, some of which are presented in chapters 4 through 7.

# Chapter 3

# Open Data Kit Sensors Framework

Smartphones can connect to a variety of external sensors over wired and wireless channels. However, ensuring proper device interaction can be burdensome, especially when a single application needs to integrate with a number of sensors that use different communication channels and data formats. Chapter 2 covered our work on hardware interfacing boards that allow sensors with low-level interfaces to communicate with mobile devices. This chapter presents Open Data Kit (ODK) Sensors, an Android framework we built to simplify the software development of mobile sensing systems. ODK Sensors simplifies both application and device driver development by providing abstractions that separate responsibilities between the user application, sensor framework, and device driver. These abstractions facilitate a component-based framework that lets developers focus on writing minimal pieces of sensor-specific code, which enables the creation of an ecosystem of reusable sensor drivers. This work provides evidence for contribution 2 of the thesis: **A software framework to simplify development of mobile sensing applications.**

This chapter starts by exploring the need for a sensing framework on mobile devices. Section 3.2 describes how ODK Sensors fits within the ODK tool suite. The framework and three alternative application-level driver architectures are discussed in Section 3.3. The three architectures helped us understand the trade-offs in performance, device portability, simplicity, and deployment ease in the context of four developing world sensing applications, summarized in Section 3.4. These applications highlight a range of sensor usage models that vary in data types, configuration methods, communication channels, and sampling rates to demonstrate the framework's effectiveness. In-lab experiments to evaluate the framework and driver architectures are presented in Section 3.5. Section 3.6 discusses future directions of this work, 3.7 describes others' related work, and 3.8 concludes this chapter.

## 3.1 Motivation

Market penetration of smartphones as a computing and communications platform has increased significantly in recent years [41]. Basic feature phones are gradually being replaced by relatively inexpensive smartphones in developing countries. For example, in Kenya, the Android based Huawei IDEOS is sold for approximately $80 USD [22]. Researchers and practitioners in the information and communication technologies for development (ICTD) community are increasingly leveraging smartphones to improve information management in under-resourced environments. Our work is motivated by the platform shift from traditional PCs and standalone sensing appliances to mobile devices (e.g., smartphones, tablets) coupled with cloud services to create mobile information systems. There is an unprecedented opportunity to integrate consumer mobile devices with external sensors, enabling the collection of data directly on these devices.

Unlike traditional personal computing devices, the new consumer devices are locked by service providers or manufacturers, and most end-users do not have the administrative rights, technical ability, or organizational capacity to modify or customize the operating system (e.g., install device drivers). As a result, relying on conventional in-kernel device driver frameworks to integrate external sensors with consumer smartphones is impractical. The ODK Sensors framework explores ways to package software so that non-technical users can access external sensors from a locked mobile device running a stock version of the Android operating system. The framework assumes the consumer device is 'locked down' and that an end-user has the skills only to install applications from a standard app marketplace such as Google Play (Google's Android app store).

The ICTD community has begun investigating the use of sensor-based systems to perform in-situ and remote monitoring (e.g., [11], [42], [43]). Even though capturing sensor data directly eliminates many of the errors that plague traditional data collection techniques, such as manual form-filling, it is still not widely used in developing regions because of the high level of technical expertise required to develop a mobile sensing application. The technical challenges include managing different physical communication channels, processing sensor-specific data, developing a user interface and designing application control logic. Unfortunately, this level of expertise is often not readily available in developing regions or even in developed regions on projects undertaken by resource-limited organizations (such as non-profits and community

groups). These complications pose technical barriers that, if reduced, would enable more applications to leverage sensors for data collection across varied domains.

The ODK Sensors framework aims to lower these barriers by simplifying the development and deployment of mobile applications that use sensors. More specifically, the goals of this work were:

1. To create a modular framework for adding new sensors by abstracting away the management of discovery, communication channels, and data buffers. Integrating a new sensor should require adding only its data handling and configuration primitives.
2. To provide a high-degree of isolation between applications and sensor-specific code. Applications should continue to function even if sensor-specific code is buggy or a sensor stops working.
3. To facilitate the integration of new sensors into applications by making it possible to download new sensor capabilities from an application market rather than requiring modifications to the OS configuration.

The ODK Sensors framework provides a single sensing interface for both built-in and external sensors. Having a single interface is appropriate for lightly trained technical workers because it hides many technical details involved in developing sensing applications. The framework also provides a simple, high-performing, and flexible abstraction on which to develop and deploy user-level device drivers on Android. While a device driver abstraction is a standard concept, the framework includes features that make development of device drivers easier by handling sensor state (e.g., connection, buffered data, threading) and requiring driver developers to implement only sensor-specific commands and data processing.

## 3.2 ODK and ODK Sensors

Open Data Kit [1] is a successful suite of mobile tools that exploit the rich interaction and high-performance computing capabilities of smartphones to improve information collection, distribution, and decision-support. It focuses on deployment contexts where conventional computing solutions (i.e., informed by concerns of the developed world) are often inappropriate due to constraints such as affordability, infrastructure, institutional capacity, and technical support. ODK Sensors expands ODK by creating a framework to ease the augmentation of a

mobile consumer device with sensing capabilities. The ODK Sensors framework supports a variety of external sensors that vary by the type of data they collect, the communication channel over which they interact with the mobile device, and the rate at which they generate data. It provides a unified interface for sensing on Android devices by combining both built-in and external sensors into a single interface. While this design maximizes the variety of sensors available through a uniform interface, the gains in ease of development are more significant for external sensors that require more programming to interface compared to built-in sensors.

The ODK Sensors framework reduces the complexity of building sensor-based mobile applications by providing abstractions that encapsulate communication channels in addition to delineating user-application functionality from sensor communication. The framework has three users: application users (end-users), application developers, and sensor driver developers. A typical application user is assumed to be the least technically proficient of the three and is expected only to be able to use applications on an Android device. An application developer is expected to know how to create new Android applications (design UIs and implement application domain logic) but is not expected to have detailed knowledge of sensor control specifics or how sensors represent and communicate their data. A sensor driver developer is the only user expected to understand the low-level protocol used by a specific sensor for configuration and data packaging but is not expected to deal with communication channel setup or multiplexing. The segregation of application, driver and framework code leads to a clean separation of developer roles and lets the application developer focus on higher level, application-specific concepts and the driver developer focus on creating sensor-specific drivers.

ODK Sensors shifts responsibilities to the framework's developers to simplify the creation of sensing applications while maintaining a high-level of flexibility for integrating new sensor types. To encourage new driver development, the framework assumes responsibility for aspects common to many sensors, including management of connection state and threads. Additionally, decomposing the system into modules enables more effective testing and code reuse, thereby improving overall system robustness; this is particularly important for ICTD deployment settings (since systems deployed in remote locations are logistically difficult – in terms of costs, time, and complexity – to update in the field).

For the framework to successfully enable an ecosystem of external sensors, it must be:

1. Easy to create sensor drivers, minimizing the knowledge and amount of code required to create a driver
2. Easy to integrate/reuse external sensors in a wide variety of applications
3. Easy to deploy the framework and device drivers, shielding an end-user from the technical details of the sensing infrastructure
4. Easy to upgrade the framework and sensor drivers
5. Difficult for bad driver code to damage the framework
6. Easy for end-users to discover available sensors through a streamlined user interface
7. Easy to manage communication channel details, such as proper handling of dropped connections

ODK Sensors attempts to meet these requirements by creating an environment of reusable components for the development of mobile sensing applications.

## 3.3 Framework

ODK Sensors simplifies the development of sensor-based mobile applications by creating a common abstraction point that enables all sensors to be accessed through a unified interface. Creating a single-interface reduces complexity since all external sensors, as well as Android's built-in sensors, are exposed through a common interface regardless of the communication medium used. The interface encapsulates communication and delineates user-application code from sensor-specific driver code, freeing application developers from having to understand the specifics of the underlying communication between an Android device and an external sensor.

From an end-user's perspective, the overall architecture for ODK Sensors consists of three apps: the *User-Application App,* the *ODK Sensors Framework App,* and the *Sensor Driver App*. In this chapter, an Android application (software downloaded from a market) is referred to as an "app", whereas the word "application" refers to usage/deployment examples. The *ODK Sensors Framework App* manages low-level, channel-specific communications and provides abstractions to isolate sensor driver code. The *User-Application App* communicates with sensors through the unifying framework API (explained in Section 3.3.1). Figure 17 shows an end-user view of ODK Sensors on a smartphone. In the figure, two apps (*User-Application App* and *ODK Sensors Framework App*), a *USB Bridge*, and a temperature probe are used to monitor the

pasteurization of milk to eliminate contaminants (e.g. HIV) in breast milk (described in Section 3.4).

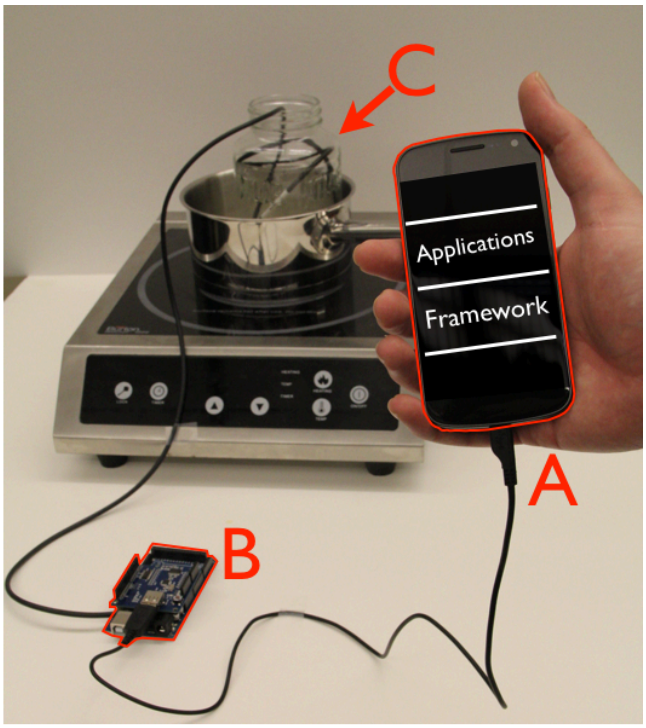



**Figure 17: End-user view of a smartphone using the ODK Sensors framework to connect to a temperature sensor via an Arduino *USB Bridge* for the milk pasteurization application (cf. Section 3.4). *Applications* and *Framework* are Android apps that are installed on the mobile device (A). The mobile device is connected to an Arduino interfacing board (B) over USB, forming a *USB Bridge* to the temperature sensor (C) using the Arduino board's I/O ports (B). The *Application* uses the *Framework* to get data from the temperature sensor over USB via the ODK Sensors API.**

| Android Construct | Description |
|---|---|
| **Service** | Runs in main thread of hosting process. Requests from other processes are handled concurrently by a threadpool. |
| **Broadcast Receiver** | Asynchronous broadcast receiver. It receives broadcasts sent via Intents from other processes. |
| **Intent** | Asynchronous messages that represent an operation to be performed, such as broadcasts, start services, start application, etc. |
| **Bundle** | Mapping from String values to parcel-able types. A *parcel* is a data container used for IPC. |
| **Manifest** | Provides essential information about the application (permissions, package name, etc.) to the Android OS. |
| **Content Provider** | Stores and retrieves data that is accessible by all applications. |

**Table 3: Android constructs used in the three framework implementations. The constructs listed are used for application metadata, packaging, storing, or sending data between applications, and forming the basis of inter-process communication with *User-Application Apps* and sensor drivers in V2 and V3 (Section 3.3.2).**

We chose Android as the target platform for the ODK Sensors framework because it is open-source, has extensive support for background processes, and includes several built-in constructs for inter-process communication (IPC) between Android applications. Examples of these Android constructs are detailed in Table 3. For instance, a *Broadcast Receiver* uses non-blocking message passing to communicate between applications, whereas a *Service* construct communicates through a blocking IPC mechanism. The ODK Sensors framework uses the Android Interface Definition Language (AIDL) to specify the programming interface that both the client and service use to communicate. From the AIDL, Android generates IPC code to decompose objects into primitives that the operating system can marshal as *parcels* across process boundaries to provide blocking IPC functionality. These constructs enable a comparison between a single-threaded asynchronous model (*Broadcast Receiver*) and a multi-threaded synchronous model (*Service*) for inter-application communication. Additionally, Android supports multiple communication APIs that facilitate connection to a wide variety of external sensors. While APIs to access Bluetooth and Wi-Fi networks have long been available on Android devices, in 2011 Android added USB support through the Android Open Accessory (AOA) Protocol [44], enabling compliant devices to connect to external hardware over USB. To include varied external sensors in the framework, we utilize *Bridges*; these are hardware-interfacing boards that connect the Android to sensors that use diverse, low-level I/O protocols (e.g. $I^2C$, SPI). This work is discussed in Chapter 2.

ODK Sensors presents a *Unified Framework Interface* to all top-level user applications (see Figure 18). User-Application apps need only implement the application-specific logic that handles processed sensor data received from the framework. For each call into the framework, the *Sensor Manager* dispatches the commands to the appropriate sensor object that, in turn, utilizes a sensor driver to perform specific low-level tasks. The framework supports multiple communication modalities by providing abstractions called *Channel Managers,* which encapsulate complexities specific to each communication channel. ODK Sensors supports multiple data types, sample sizes, sampling frequencies, and sensor configurations by utilizing *Sensor Driver* abstractions that encapsulate sensor-specific data processing. These abstractions let applications interface with sensors using higher level, key-value pair constructs that are not constrained to be fixed-size arrays or values of a specific type. This lets application developers focus on application logic instead of sensor-specific logic.

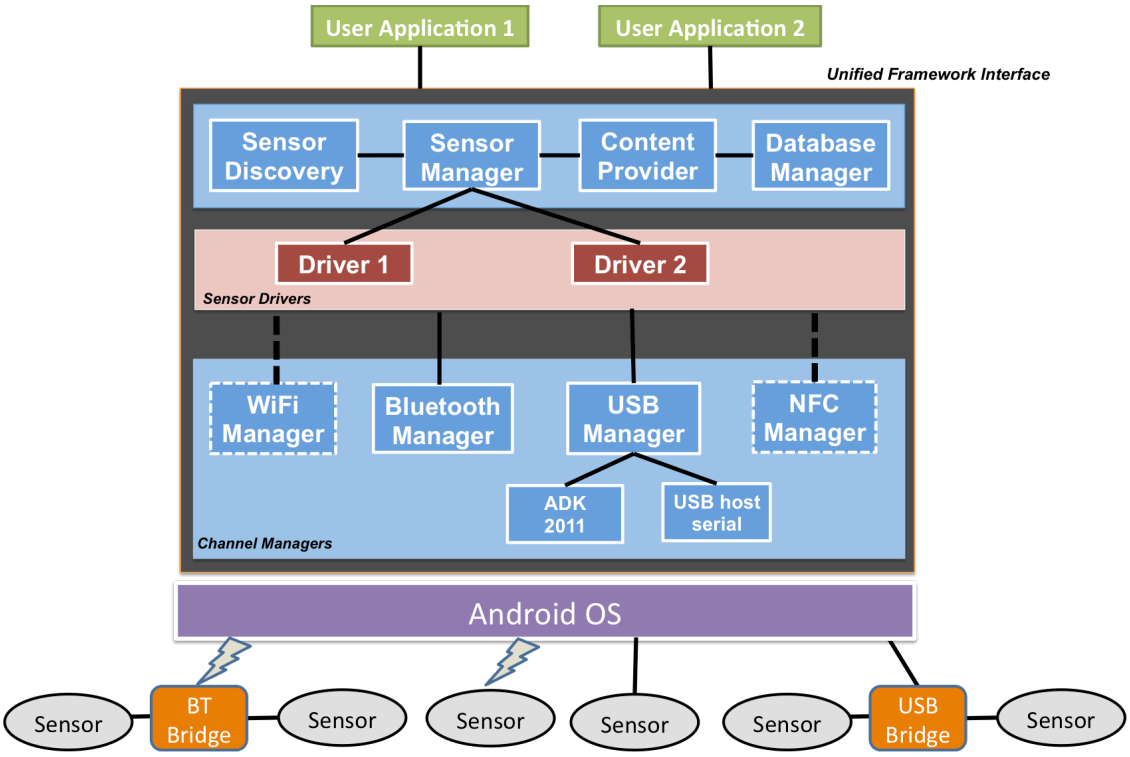



**Figure 18: Architecture overview of the ODK Sensors system. The Unified Framework Interface is the common access point for applications to interact with the framework. Sensor Manager maintains references to all sensors and the corresponding sensor drivers. Channel Managers handle connections over the communication channels (e.g., USB, Bluetooth). The WiFi and NFC Channel Managers (with the dotted box outline) will be implemented in the near future. Note: All architectures use the same data flows; however, V2 and V3 sensor drivers move to separate apps, as described in Section 3.3.2.**

The framework's communication subsystems provide abstractions for lower-level, channel-specific communication protocols that make it easier for a sensor driver developer to interface with an external sensor. The framework encapsulates communication channel specifics within the respective channel managers to hide them from application and sensor driver developers. For instance, Bluetooth-enabled sensors must be discovered and paired with the Android, and a socket must be set up for communication. ODK Sensors automatically manages this entire process for the application developer. The current implementation supports communications over Bluetooth and USB; in the future, we plan to add additional channel managers to support other communication methods, such as NFC and WiFi. The USB channel manager supports two types of USB devices: (1) devices that implement the AOA protocol, and (2) USB slave devices accessed over a serial protocol (e.g., FTDI) from Androids that have USB Host capability. To create a single unified sensing interface, the ODK Sensors framework also

exposes all 11 built-in Android sensors (for Android 2.3 and greater), thus creating a single integration point for collecting sensor readings.

A *Sensor Driver* handles the particular messaging protocol that configures and/or requests data from an external sensor by issuing commands to the appropriate *Channel Manager*. During data collection, the *Communication Manager* passes all raw data received from the sensor to the appropriate sensor driver via the *Sensor Manager*. Sensor drivers receive and process data encoded in formats specific to their respective sensors and generate configuration commands as required by the sensor. The sensor driver parses this sensor-specific data, transforming it into key-value pairs that can be easily consumed by top-level user applications. Likewise, the sensor's configuration parameters are specified as key-value pairs by user applications and passed to the appropriate *Sensor Driver* by the *Sensor Manager*. The driver encodes these key-value pairs according to the sensor's messaging protocol, and the encoded data is sent to the sensor via the *Channel Manager*. By having channel managers handle communication details, driver developers no longer need to be aware of channel-specific protocols. Instead, they can simply implement the interface described in Section 3.3.2 and convert raw sensor data coming from the communication channel into key-value pairs, and vice versa for sensor configuration data.

In addition to allowing for easy reuse, the *Sensor Driver* design shields applications from changes in the communication protocol, configuration, or data type. This leads to more robust systems that are easier to maintain. The sensor driver abstraction also enables multiple apps to interface with the same type of sensor by reusing an existing *Sensor Driver*. The framework's sensor drivers are designed as stateless processors of data to shield driver developers from tasks required to enable interaction with multiple identical sensors simultaneously (e.g., channel management, threads, buffers). To better understand which Android IPC constructs to use to create our application-level (or user-level) sensor driver framework, we implemented three different versions of the framework. The *Sensor Manager* and *Channel Managers* communicate with sensor drivers either with method calls inside the framework (V1), with IPCs through a Service Interface (V2), or with broadcasts to Broadcast Receivers (V3). To establish a baseline, version 1 of the framework was created as a single app. Details of the interfaces presented to developers and the framework's communication subsystems are discussed in the following subsections.

### 3.3.1 Unified Framework Interface

The *Unified Framework Interface* exposed by ODK Sensors abstracts many sensor-specific details and hides channel-specific quirks. This interface provides a common entry-point for user applications, allowing them to leverage both built-in sensors and external sensors connected over different communication channels. The code snippet below shows methods of the ODKSensorService, an Android Service that is part of this interface.

```
interface ODKSensorService {
  boolean sensorConnect(in String id, Boolean useContentProvider);
  void configure(in String id, in Bundle config);
  boolean startSensor(in String id);
  boolean stopSensor(in String id);
  List<Bundle> getSensorData(in String id, long maxNumReadings);
  boolean isConnected(in String id);
  boolean isBusy(in String id);
  boolean hasSensor(in String id);
}
```

Each method in the ODKSensorService requires the sensor's ID to control a specific sensor in the framework. If applications do not know the Sensor ID, then the framework provides an interactive discovery process for users, freeing the application from implementing its own sensor discovery mechanism. To launch the ODK Sensors' interactive sensor discovery UI, the application simply sends an Intent to the framework, as described in Section 3.3.2.4. Once the application has the sensor ID, it can connect to the sensor and begin retrieving sensor data by periodically calling the `getSensorData` method of the Service. This method returns sensor readings to the application in a mapping of key-value pairs created by the *Sensor Driver'*s processing of raw incoming sensor data. As an example, a temperature sensor's driver parses data according to the messaging protocol of the sensor to extract multiple (or single) temperature readings in degrees Celsius and passes a list of Android Bundles that map key = "temperature" to value = "value in °C" to the application. The framework expects the application to retrieve data in a timely fashion to clear the memory that is buffering the data. In cases where the application does not plan to read the data immediately, the framework provides a second mechanism to applications for retrieving sensor data stored in a local database through a *Content Provider*. Applications specify whether the framework should put the data in a database to be accessed by a Content Provider when calling `sensorConnect`. An application can query the Content Provider whenever it wants to get data from a sensor that the framework considers to be owned by the application.

### 3.3.2 Sensor Drivers

Sensor drivers are designed to abstract sensor-specific control code from more general sensor management code. Certain concepts are common to all sensors, such as initialization, configuration, and taking readings, but the framework need not know specifics of how each sensor accomplishes these tasks. Sensor drivers enable the framework to communicate with sensors while maintaining the necessary abstractions that keep the framework modular and extensible. The same driver interface is used for all sensors – both built-in and external sensors: the driver interface abstraction encapsulates sensor-specific data transfer and processing and hides sensor specifics, such as data types, frequency of collection, data size, and various configuration parameters. The driver abstractions let the framework reuse core functionality, such as sensor configuration, connection, communication handshakes, buffering data, multiplexing, etc., for multiple types of applications. Simultaneous integration of different sensors involves complex tasks – such as concurrent Bluetooth and USB setup – that requires multiple data sockets and threads to buffer and process the data from these connections. The framework hides these complexities, thereby significantly simplifying the job of both the application developer and the sensor driver developer.

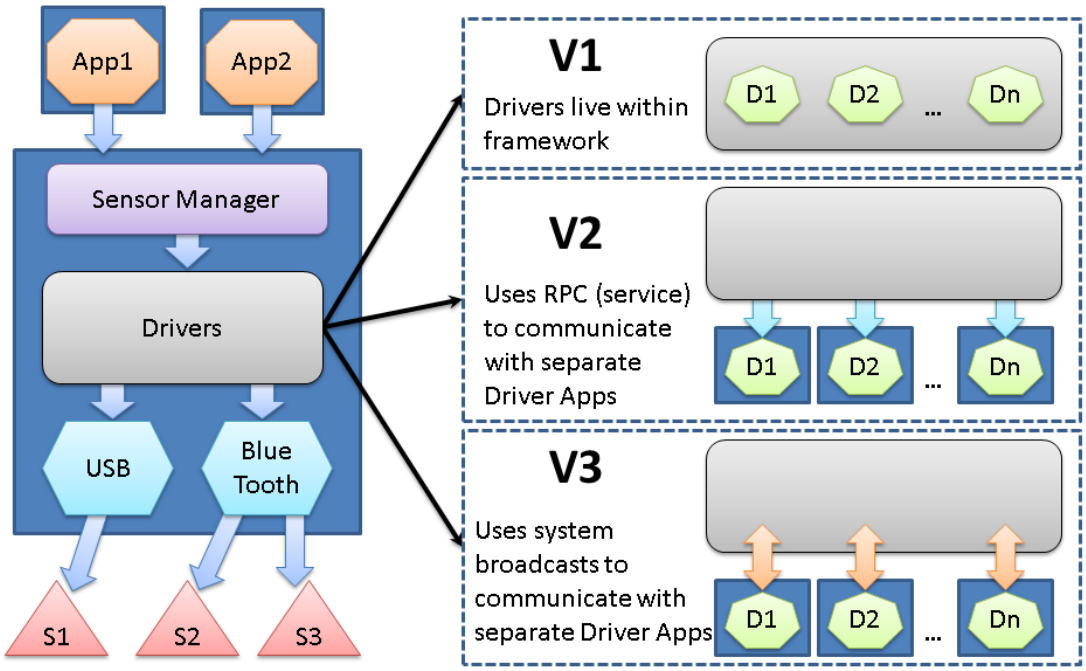



**Figure 19: The three different framework implementations that vary with respect to how the framework communicates with sensor drivers. Each dark blue rectangle represents a separate application. The design in which the drivers live within the framework is referred to as V1. The V2 design uses remote service calls to communicate with separate driver apps outside the framework. V3 uses system broadcasts to communicate with these separate driver apps.**

Moving sensor drivers into separate Android apps that implement a common driver interface improves the framework's modularity and extensibility. Separating drivers from the framework lets drivers for new sensors be downloaded and installed onto an Android device from any Android marketplace or website, like any other Android app. To understand the various architectural trade-offs of moving sensor drivers to external Android apps, three different framework architectures were created to compare two different Android IPC mechanisms. As depicted in Figure 19, one design keeps sensor drivers within the framework (V1) and serves as a baseline, and two designs move each driver to its own external Android app (V2 & V3). To understand the best way to communicate with the external drivers, one framework used a blocking Binder IPC call (V2), while the other used a non-blocking message passing structure (V3). Both of these versions let end-users dynamically add drivers to the ODK Sensors framework as needed. User applications remained the same across the three implementations, since their interface with the framework did not change. In each framework implementation, every driver implements the following interface:

```
public interface Driver {
  byte[] configureCmd(Bundle config);
  byte[] sendDataToSensor(Bundle data);
  SensorDataParseResponse getSensorData(long maxNumReadings,
  List<SensorDataPacket> rawSensorData, byte [] remainingData);
  byte[] startCmd();
  byte[] stopCmd();
}
```

Drivers implement this interface to specify how to parse raw sensor data and what, if any, sensor-specific messages to send for configuration, getting data, and starting or stopping the sensor. As noted, the device drivers are stateless processors of data, which means the programmer need not manage multiple sensor instances. Methods such as `getSensorData` require the sensor driver to maintain access to buffered data it has previously processed as well as raw sensor data it has not yet processed. To facilitate this, `getSensorData` returns a `SensorDataParseResponse`, which is simply a list of key-value pairs that have been fully parsed as well as any leftover bytes from the input stream. We eliminate state from the driver itself; instead, we have the framework handle all sensor state, including buffered data, and provide it to the driver at the appropriate time when the application requests data. Implementing an external sensor driver for each of the three architectures differs slightly, as described in the following subsections. Built-in sensor drivers are included with the framework even though it is

possible to implement them as separate apps since communication is handled internally by the Android OS. The commands are Android-specific, not communication-channel specific, so it seems unnecessary to move them outside of the framework.

#### *3.3.2.1 V1: Drivers within the Framework*

In the first version, V1, we placed all sensor drivers inside the framework, thus creating a single Android app. At runtime, the framework accesses individual sensor drivers from an in-memory map. In this architecture, the drivers, by executing within the framework, create a tight coupling that should have the best performance and provide a baseline against which to compare the other versions. However, this design is not ideal for end-users to dynamically add new drivers since it requires a recompilation or the use of a class loader. Using a class loader is not ideal for a population of non-technical users; it requires placing files into proper access-controlled directories on the device, thereby enabling the framework to dynamically load classes. Our goal is to build a system that uses established Android distribution and communication mechanisms to dynamically deliver and add drivers to the framework.

#### *3.3.2.2 V2: Driver Communication via Services*

In the V2 version of the framework, sensor drivers were implemented as separate Android apps. Each app implements an Android *Service* that defines the `Driver` Interface using AIDL, enabling the framework to communicate with the sensor driver via Android's Binder IPC. Android provides an AIDL to define the programming interface between the client and server and automatically generates stub Binder IPC classes. When the client calls these methods, the system copies the payload from the client into kernel memory, which is memory-mapped into the server's address space. The server-side procedure then handles the call. The sensor driver application includes driver-specific metadata that is used by the framework for driver discovery (described in Section 3.3.2.4). If an appropriate driver application is installed on the device, a generic sensor object is constructed to act as a proxy between the driver and framework by binding to the `Driver` interface, which is presented as an Android Service. The framework maintains a reference to the specified generic sensor object that communicates with the driver using Android's Binder IPC. Each driver proxy acts as a thin wrapper for the driver and contains a reference to the appropriate channel manager. Communication with the sensor forwards commands returned from the driver so that they are transmitted over the appropriate channel.

Since the driver application can be reused by multiple instances of the same type of sensor, it is important that it be stateless. Therefore, the driver's remote function calls are designed to transfer the required state to the driver on each service call and allow the driver to return state information along with the parsed data. An example of state information stored by the framework is the excess bytes from the input data stream, enabling buffering of the input stream until enough data is received to parse and produce a full message.

#### *3.3.2.3 V3: Driver Communication via Broadcasts*

The third and final version of the framework, V3, also implements sensor drivers as separate Android applications but uses message passing with *Broadcast Receivers* for IPC (instead of synchronous blocking IPC). Each driver specifies a unique broadcast address that is discovered by the framework during the driver discovery phase (described in Section 3.3.2.4). The framework communicates with the driver by sending messages to the driver's unique broadcast address. Included in the broadcast message sent to the driver is a unique broadcast address for the driver to use to send a response back to the framework. The additional information included in each of the *Intents* (i.e., the data exchanged) is part of the API between the framework and drivers. By instantiating a broadcast receiver for each instance of a sensor, the framework can multiplex responses for each individual sensor. This API provides the same functionality as the interface implemented by sensor drivers in V1 and V2. The message-passing interface is designed to communicate state between the framework and the sensor driver, enabling the sensor driver to cache incomplete information between processing sensor readings.

This architecture decouples the drivers from the thread of control, allowing a non-blocking message passing framework, but it incurs the overhead of using Android *Broadcasts* for data communication. The framework is better shielded from buggy drivers due to the decoupling enabled by the blocking semantics. However, we acknowledge that broadcasts can be intercepted by another Android application, which poses a security risk.

#### *3.3.2.4 Sensor and Driver Discovery*

Each version of the framework semi-automates the sensor discovery process by providing a pre-built UI that application developers can launch when they need the user to select one of the currently available sensors. When the framework gets a request for an unknown sensor, it informs the user's application to launch the framework's built-in sensor discovery system to find and pair the appropriate device driver for the desired sensor. The user simply needs to select the

sensor along with the corresponding driver from the list presented. The framework displays a list of available drivers for the relevant communication channel, as well as a list of sensors that are already physically connected to or in Bluetooth wireless range of the device. In V2 and V3, the framework automatically discovers installed driver applications by searching Android *manifest* files. Once the user has mapped the sensor with the appropriate sensor driver, the framework does the rest, and the sensor ID is returned to the application. This ID lets the application control the sensor with the methods presented in the ODKSensorService Interface (Section 3.3.1). Once a sensor has been discovered, it is stored in the *Sensor Manager'*s database, which lets the application skip the discovery step in the future.

Having sensor drivers as separate applications simplifies driver distribution. Ideally, we envision a scenario where manufacturers post their drivers on their own websites or in an Android market (such as Google Play). By taking advantage of Google's standard application distribution system, the framework lowers barriers for novice users by using familiar application delivery channels. Additionally, if the device manufacturer needs to update their driver, they simply need to post the updated application to market, and the built-in Android application update system takes care of the rest. Another advantage of separating the device drivers into separate Android apps is that it gives manufacturers the opportunity to create drivers while keeping their protocols proprietary.

## 3.4 Applications

We developed sample applications to demonstrate reuse, flexibility, and extensibility of the framework. These applications exemplify three basic dimensions of variation in sensing applications: *communication channel*, *sensor configuration,* and *data collection style*, all of which must be supported flexibly by the framework. First, the channel used to communicate with the mobile device can vary across sensors and applications (e.g., USB, Bluetooth, NFC). Second, sensors have different configuration parameters or settings that need to be specified, such as sampling rate, trigger conditions or alerts, identifiers, and calibration. Third, the data needed by an application can change in format, size, and frequency of collection. The framework supports any combination these three factors transparent to the top-level user application. If a factor changes, the sensor driver requires only minimal adjustments to parse a new type of data or specify a new channel manager.

| Application | Comm. Channel | Configuration | Data Style |
|---|---|---|---|
| **Medical** | Bluetooth | - Calibrate | Single Reading |
| **MilkBank** | USB, Bluetooth | - Sampling Rate | Real-Time Time-Series |
| **Vaccine** | USB, Bluetooth | - Alerts<br>- Sampling Rate<br>- Snapshot Size | Snapshot Time-Series |
| **Time-use** | Bluetooth | - Identifier<br>- Calibrate | Historical Time-Series |

**Table 4: Variations in the example sensing applications. The MilkBank and Vaccine applications were initially implemented using a *USB Bridge* and then re-implemented using a *Bluetooth Bridge.***

Developing these applications for the ODK Sensors framework helped to evaluate whether we could reuse the abstractions for different use cases. The applications exercise both the wired and wireless subsystems of the framework and, in our experience, exemplify four commonly used modes of data collection in sensor-based systems (Table 4):

- *Single Reading*: The user requests data from the sensor and records data points by taking a single reading from a real-time stream of data. A clinician's application that connects tools (e.g., blood pressure, pulse oximetry) to a mobile device is an example of this use case. mPneumonia is one such application (cf. Section 7.4).
- *Real-Time Time-Series*: The user of a data collection node has an active session with the sensor and observes a stream of samples from the sensor in real-time. Monitoring the temperature curve of a milk pasteurization procedure (cf. Chapter 5) is an example of this use case. The temperature curve is also saved so that it can be reviewed later.
- *Snapshot Time-Series*: Sensors are deployed to autonomously monitor certain phenomena. They aggregate readings internally over a period of time and may report readings to a remote location periodically (e.g., after an alert to detect a specific condition). Temperature and electrical current sensors deployed to monitor vaccine refrigerators are an example of this use case (cf. Chapter 4).
- *Historical Time-Series*: Sensors are deployed to autonomously monitor certain phenomena (e.g., movement of an object, such as a water can, over a period of time). However, unlike a *Snapshot Time-Series*, data retrieval is not automatic and requires someone to be within range of the sensor to offload the sensor's stored data. The Time-use monitoring application

(cf. Chapter 6) exemplifies this approach and is dependent on field calibration for configuring the sensor with sampling rates and identifiers.

Three of these applications had already been deployed in developing regions as part of pilot studies. Rewriting these applications to leverage ODK Sensors simplified them significantly by separating the application logic from the communication logic. Figure 20 shows how these applications interact within the V1 framework. Chapter 7 describes other applications being developed using ODK Sensors; they also use one of the four data collection modes listed above.

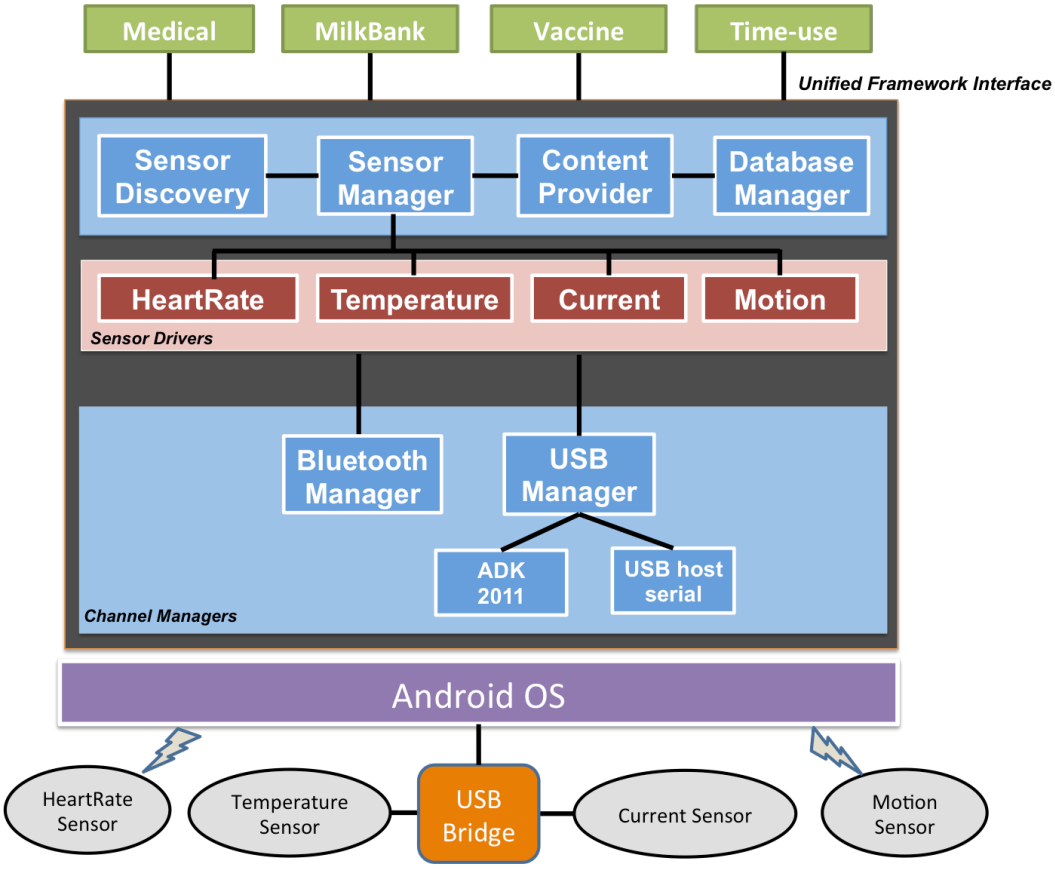



**Figure 20: V1 framework architecture incorporating the four applications discussed in Section 3.4.**

## 3.5 Experiments and Evaluation

Our experiments to evaluate ODK Sensors were designed to explore the trade-offs between the three frameworks in terms of performance, power consumption, and ease of programming. Each of the architectures was benchmarked based on the same set of applications, sensor drivers, and communication channels. The code for the framework and drivers is identical for V1, V2, and V3 with the exception of the protocol the framework uses to communicate with sensor drivers. The protocol changes required changes to four Java files. User applications were

not affected across the three implementations as their interface with the framework did not change.

### 3.5.1 Performance

Performance experiments were conducted using several different Android devices to understand the impact of hardware variability. The Samsung Galaxy Tab and Samsung Nexus S represented high-tier Android devices, the HTC Nexus One and Motorola Droid represented mid-tier devices, and the Huawei IDEOS represented low-tier Android devices likely to be common in developing regions. An Arduino Mega2560 with USB Host capability connected to a Motorola Xoom tablet was used to characterize the *USB Bridge*.

#### *3.5.1.1 Sensor Application Throughput*

First, we measured the throughput of the four applications described in Section 3.4. Each test consisted of 60 seconds of continuous data collection. The results (Table 5) show that the rate at which an application gets sensor data was often on the order of one sample every 1 to 2 seconds. In fact, three of the applications collected a data sample no more than twice per second; this rate was actually limited by the sensor itself. The Vaccine and MilkBank applications used a digital temperature sensor that required a 750ms delay in sampling the ADC, while the heart rate sensor used in the Medical application transmitted a packet every 2 seconds. Framework throughput was substantially higher than the saturation point for these three applications. However, in the Time-use application, which required a one-time bulk reading of historical data, the sensor sent its collected data as fast as possible over Bluetooth and achieved a throughput of 51 packets per second. This was close to the maximum throughput achieved on the Bluetooth channel using the framework (see Table 6).

| Application | Throughput |
|---|---|
| **Time-use (Bluetooth)** | 51.0 |
| **Medical (Bluetooth)** | 1.5 |
| **Vaccine (USB)** | 1.0 |
| **MilkBank (USB)** | 1.0 |

**Table 5: Observed data throughput (packets/sec) for four applications to receive sensor data from the framework. Packet size is not consistent across applications as the size of data samples varies.**

#### *3.5.1.2 Communication Channel Throughput*

Next, we systematically tested the saturation point of the Bluetooth and USB communication channels with a stress test that sent fake data as fast as possible over the

channels. To do this, we programmed a "spammer" sensor on two Arduino microprocessors, one that emulated a Bluetooth-based sensor and another that emulated a USB-based sensor. Each "spammer" sensor reacted to a "start" signal by executing a loop to send 1 byte data packets as fast as possible until it received a "stop" signal. Our tests allowed the "spammer" sensor to send a rapid, constant stream of data for ten minutes on each of our three framework architectures on Bluetooth and on USB. The maximum throughput that could be achieved with full saturation of the communication channels is shown in Table 6.

| Framework | Bluetooth | USB |
|---|---|---|
| **Drivers in Framework** | 58.0 | 42.5 |
| **Drivers w/ Service** | 58.7 | 41.5 |
| **Drivers w/ Broadcasts** | 49.0 | 42.0 |

**Table 6: Throughput (packets/sec) of Bluetooth and USB Channels for the different framework architectures.**

The throughput results were surprising as the USB channel had lower throughput than the Bluetooth channel. Investigating this issue further revealed that the USB Host driver on the Arduino added some delays that affected channel performance. We found that increasing the payload size did not significantly reduce the packets per second rate. After increasing the message size to 1KB per packet and disabling our reliability system, the *USB Bridge* gave a channel bandwidth of over 40KB per second. Increasing the payload size beyond 1KB was problematic and was primarily limited by the Arduino's memory size. This is acceptable for now because our real-world USB applications require throughput that is significantly lower than what we can already achieve with our current *USB Bridge*. However, in the future, we will explore alternative options for the *USB Bridge* to achieve higher performance to support more demanding sensing applications, such as those streaming high-resolution camera data.

#### *3.5.1.3 Framework Throughput*

We evaluated the performance of each framework by establishing an emulated communication channel for fake sensors to use with varying send rates and packet sizes. These experiments tested the throughput of the three framework versions by eliminating the limits imposed by real communication channels. For a baseline understanding, we evaluated the framework's throughput by varying packet size and the delay between packets (Table 7). To understand the effects of multiple sensors, we ran tests that varied the number of sensors communicating simultaneously through the framework (Table 8). Finally, we verified that the framework's performance did not significantly degrade when operating on different classes of

Android devices (high-tier, mid-tier and low-tier) (Table 9). This is important because users in developing regions will likely have a diverse set of devices.

| ***V1*** | **(bytes) 1** | **10** | **100** | **1K** | **10K** | **100K** | **Max** |
|---|---|---|---|---|---|---|---|
| **(ms)** | | | | | | | |
| **1** | 798.1 | 796.9 | 790.2 | 747.9 | Error | Error | 1000.0 |
| **2** | 441.0 | 440.8 | 438.9 | 423.6 | Error | Error | 500.0 |
| **4** | 234.6 | 234.5 | 234.8 | 229.6 | Error | Error | 250.0 |
| **8** | 121.4 | 121.4 | 121.3 | 119.7 | 111.5 | Error | 125.0 |
| **16** | 61.6 | 61.6 | 61.5 | 60.9 | 58.1 | Error | 62.5 |
| **32** | 31.0 | 31.0 | 31.0 | 30.9 | 30.1 | Error | 31.3 |
| **64** | 15.5 | 15.5 | 15.5 | 15.5 | 15.3 | 13.8 | 15.6 |
| **128** | 7.8 | 7.8 | 7.8 | 7.8 | 7.7 | 7.3 | 7.8 |
| ***V2*** | **(bytes) 1** | **10** | **100** | **1K** | **10K** | **100K** | **Max** |
| **(ms)** | | | | | | | |
| **1** | 785.5 | 786.8 | 781.1 | Error | Error | Error | 1000.0 |
| **2** | 437.6 | 438.0 | 436.0 | 421.8 | Error | Error | 500.0 |
| **4** | 233.7 | 233.8 | 233.0 | 229.0 | Error | Error | 250.0 |
| **8** | 120.6 | 120.6 | 120.3 | 119.6 | Error | Error | 125.0 |
| **16** | 61.4 | 61.4 | 61.3 | 61.0 | 58.8 | Error | 62.5 |
| **32** | 31.0 | 31.0 | 31.0 | 30.9 | 30.1 | Error | 31.3 |
| **64** | 15.5 | 15.5 | 15.5 | 15.5 | 15.3 | Error | 15.6 |
| **128** | 7.8 | 7.8 | 7.8 | 7.8 | 7.7 | 7.3 | 7.8 |
| ***V3*** | **(bytes) 1** | **10** | **100** | **1K** | **10K** | **100K** | **Max** |
| **(ms)** | | | | | | | |
| **1** | 771.5 | 768.0 | 760.5 | Error | Error | Error | 1000.0 |
| **2** | 432.2 | 431.5 | 426.5 | 419.2 | Error | Error | 500.0 |
| **4** | 231.9 | 231.9 | 231.2 | 225.2 | Error | Error | 250.0 |
| **8** | 119.2 | 119.1 | 119.2 | 118.4 | Error | Error | 125.0 |
| **16** | 60.9 | 60.8 | 60.8 | 60.4 | Error | Error | 62.5 |
| **32** | 30.8 | 30.9 | 30.8 | 30.7 | 30.1 | Error | 31.3 |
| **64** | 15.5 | 15.5 | 15.5 | 15.4 | 15.3 | Error | 15.6 |
| **128** | 7.7 | 7.7 | 7.7 | 7.7 | 7.7 | Error | 7.8 |

**Table 7: Throughput results (packets/sec) for a Nexus One when varying the packet size and inter-packet generation delay for the three framework versions. The 'Error' value indicates a test run that did not complete because of a memory error. The Max column lists the theoretical maximum throughput.**

We measured framework throughput (packets/second) by sending data at varying rates with varying sizes (Table 7). The test results reported the number of packets received on average per second for each framework on a Nexus One. Each test ran for three minutes on five different Nexus One phones. The results of the tests were averaged. The send delay values began at 1ms and were increased by doubling the delay to a max of 128ms, while the packet size started at 1 byte and was increased by an order of magnitude for each test up to a maximum of 100,000 bytes. We varied these parameters to understand any limiting factors inherent in the different IPC

mechanisms. Differences in performance of the various architectures became negligible as the send delay became the dominant limiting factor. None of the frameworks completed the most strenuous tests – 100,000-byte packets being sent with only a 1ms delay – because of memory errors that invalidated the results (indicated by "Error" in the table). Generally speaking, the throughput values of the framework were similar since the cycles spent in IPC were a small part of the total framework execution time. Where the send delay was not the dominant factor, V1 performed the best since it did not incur IPC overhead, whereas V3 had lower performance since it communicated with drivers using message passing.

The primary difference between V2 and V3 is the IPC method. Our results confirm our expectation that the broadcast intent system has more overhead than a synchronous Binder IPC call. This finding correlates with other research that showed the Android system implements the *Intent* call as two IPC-style calls – one from the sender to the system and one from the system to the receiver [45]. While the 'double call' for V3 lowers performance, broadcast communication represents only a small amount of the overall work. Therefore, while the effects are visible, they are generally negligible except at faster data generation rates. Our results also matched the findings of a drop-off in performance for large packets. The Android system allocates a default 4KB kernel-side buffer for each process that will receive data from Binder IPC calls. When the payload size exceeds 4KB, the system allocates an additional, temporary buffer to transfer the data. These larger IPC calls incur additional allocation overhead and were shown to be inefficient as the Android system memory-maps each additional page separately, instead of mapping them all at once [45]. These factors result in a performance hit for payload sizes over 4KB that is reflected in our results for packets 10KB or larger.

To compare how framework throughput (packets/second) changes when multiple sensors are simultaneously generating data, we ran tests with multiple fake sensors generating 100 byte data packets every 64ms. In this test case, drivers simply copy incoming bytes into a parsed sensor reading. The results (Table 8) show that the frameworks perform close to the theoretical max when there are a few sensors running simultaneously. Differences in framework throughput start to emerge as the number of sensors increases. However, we consider 12 sensors to be adequate for all the realistic applications we have considered to date.

| Num | V1 | V2 | V3 | Max |
|---|---|---|---|---|
| **6** | 93.3 | 93.2 | 93.1 | 93.8 |
| **12** | 186.5 | 186.4 | 185.6 | 187.5 |
| **24** | 371.9 | 371.8 | 365.8 | 375.0 |
| **48** | 737.6 | 735.2 | 686.4 | 750.0 |

**Table 8: Framework throughput (packets/sec) of multiple sensors sending 100-byte packets every 64ms on a Nexus One.**

To understand how performance would change on various Android platforms, we measured V2's performance on different models of Android devices. Like the tests in Table 8, we used 100-byte packets sent every 64ms and varied the number of sensors simultaneously moving data through the driver. As expected, the high-end devices (Samsung Nexus S, and Galaxy Tab) performed best while the IDEOS performed worst with 48 sensors concurrently running. Overall, the throughput of the V2 framework for 12 sensors is similar across the four devices, confirming that the framework should be portable to a variety of Android devices. While the results presented in Table 9 show that all devices could support multiple sensors with a send rate of 1 packet every 64ms, other tests showed that faster send rates eventually caused errors on all devices. Devices with smaller amounts of RAM and smaller default heap sizes seemed to have more issues. For instance, when delay between packets was reduced to 32ms, the IDEOS experienced out of memory errors with 48 sensors, while the Nexus S did not experience errors until 96 sensors were simultaneously sending data. Again, only minor differences were seen when considering realistic numbers of sensors.

| Num Sensors | Galaxy Tab | Nexus S | Droid | IDEOS |
|---|---|---|---|---|
| **3** | 46.4 | 46.6 | 44.4 | 46.3 |
| **6** | 92.8 | 93.2 | 92.1 | 92.2 |
| **12** | 185.0 | 186.2 | 184.0 | 183.8 |
| **24** | 368.6 | 372.5 | 366.4 | 360.6 |
| **48** | 733.7 | 741.0 | 720.4 | 677.4 |

**Table 9: Throughput (packets/sec) of V2 framework on different Android devices (100-byte packets sent every 64ms).**

### 3.5.2 Power

Power consumption for the frameworks was negligible with respect to the power use of a device with active Bluetooth or an illuminated screen. We ran 12-hour tests where each framework processed 100 packets/second on a Nexus One. At the beginning of each test, the battery was charged to 100%, and each test resulted in an approximate 3% drain of the battery. In these tests, the screen, Bluetooth, Wi-Fi, and cellular radios were off, and the only applications running were the battery monitoring application and the sensor framework. These tests show

only negligible differences between the frameworks with regard to power consumption for expected operating conditions. The tests also show that the power consumption of the framework is negligible compared to other phone components.

### 3.5.3 Ease of Use/Programming

The ODK Sensors framework provides a reusable code base that makes it easier to create sensing applications by providing a common interface for all sensors (both built-in and external) that abstracts sensor communication details. By decomposing a sensing application into discrete and reusable building blocks, we enable a plug-and-play capability where users with different technical skill levels can contribute in different ways. The system is designed to leverage app marketplaces, which provide a standard interface for users to find and download a variety of applications, making it easy to enhance their Android device to use external sensors. Both application users and developers can leverage marketplaces for distributing and locating sensing applications and sensor drivers that can be reused for a variety of mobile sensing needs.

An application user who wants to collect data from sensors can simply download the ODK Sensors framework, a sensing application (written by an application developer) and the sensor driver(s) (written by driver developers) that corresponds to the sensor(s) they require. An application developer can re-use any existing sensor driver, allowing them to write a sensing application by simply downloading the appropriate driver and interacting with it using the framework's interface. A driver developer's responsibilities are limited to implementing the sensor-specific communication requirements (e.g. configuration, start/stop commands, etc.). The driver developer can focus on sensor-specific issues and ignore communication channel specifics of the various wired and wireless channels.

A final measure of ease of programming is that the system is portable. In ODK Sensors, all Android devices are interchangeable within the framework as long as the requisite apps and drivers have been installed.

#### *3.5.3.1 Application User*

The application user is expected to be able to obtain and use basic Android applications. To make device driver distribution easy, V2 and V3 drivers are designed to be separate applications, enabling device manufacturers to post them on their websites or in an Android marketplace. By using marketplaces such as Google Play, manufacturers take advantage of

Android's application distribution system. This makes it easy for users to find and keep their driver updated, as their device will receive updates automatically (similar to Windows and Mac updates). The framework tries to minimize the work of the end-user by semi-automating the sensor discovery process. It automatically discovers all installed sensor drivers on the Android device and populates a list of available devices on each communication channel. The user simply selects the sensing device and is then prompted with a filtered list of possible corresponding drivers. The framework's interface also guides the user through any channel-specific pairing tasks (such as on Bluetooth) required to interact with the sensor.

#### *3.5.3.2 Application Developer*

The framework simplifies the application developer's role by providing a single interface to communicate with all external and built-in sensors. This significantly reduces the number of software packages an application developer must understand and use in order to communicate with different types of sensors over varied communication channels. To further minimize developer responsibilities, ODK Sensors provides a base class for application developers that encapsulates the calls to the framework's unified interface, removing the need for the developer to understand Android IPC specifics. Additionally, the framework reduces application developer responsibilities by providing a sensor discovery and driver detection user interface. If an application needs the user to discover a sensor, it can send an *Intent* to ODK Sensors to launch the framework's discovery UI. The UI guides the user through sensor discovery, pairing, registering, and driver selection and then returns the selected sensor ID to the application, thus eliminating the need for the application developer to re-implement common functionality.

| Application | Standalone App | App using Framework | Sensor Driver |
|---|---|---|---|
| **Time-use** | 1350 | 956 | 139 |
| **Medical** | 933 | 246 | 80 |
| **MilkBank** | 1325 | 316 | 105 |
| **Built-in Accelerometer** | 546 | 538 | 29 |

**Table 10: Comparison of the lines of code needed to create a standalone sensing application vs. an application that uses the ODK Sensors framework. The Sensor Driver column shows the lines of code in the framework's sensor drivers.**

To quantify the simplification of development, we ported four existing standalone sensing applications to leverage the framework (cf. Section 3.4). The reduction in the lines of code (Table 10) in the ported applications provides a basic measure of how development was

simplified. Standalone applications have larger code bases because they must implement additional logic to manage communications and handle sensor-specific data parsing.

| Module | Purpose |
|---|---|
| **Permissions** | Allow access to Bluetooth in Manifest |
| **BluetoothAdapter** | Performs basic Bluetooth tasks like device discovery and creation of BluetoothDevice and BluetoothServerSocket |
| **BluetoothDevice** | Represents a Bluetooth device to create a connection or query information from it |
| **BluetoothServerSocket** | Listens for connections and creates a BluetoothSocket to manage the connection |
| **BluetoothSocket** | Allows streaming transport over Bluetooth |
| **InputStream** | Provides input to BluetoothSocket |
| **OutputStream** | Provides output from BluetoothSocket |
| **IOException** | Deals with communication exceptions |
| **UUID** | Used for initiating an RFCOMM communication |
| **UI Components** | Makes a UI that allows a user to discover, pair and connect with a device |
| **Handler** | Receives updates from a Bluetooth I/O thread |
| **Message** | Contains data to be sent and handled with the Handler described above when Bluetooth device state changes or data is read/written |
| **IntentFilter** | Filters Intents for a BroadcastReceiver |
| **Broadcast Receiver** | Listens on BluetoothDevice state changes for when a device is discovered |

**Table 11: Additional modules used in a Bluetooth-based sensing application written without using ODK Sensors.**

The ported applications leverage the framework for channel and connection management, while sensor-specific data processing is delegated to sensor drivers. Table 11 shows the additional Android modules needed in a standalone application that interacts with sensors over Bluetooth. Table 12 shows the Android modules needed to communicate with a sensor over a *USB Bridge*. Developers leveraging the ODK Sensors interface do not need to understand these Android constructs since the framework hides module internals from application developers.

| Module | Purpose |
|---|---|
| **Permissions** | Allow access to USB in Manifest |
| **UsbManager** | Allows enumeration of and communication with connected USB accessories |
| **UsbAccessory** | Represents a USB accessory and contains methods to access its identifying information |
| **ParcelFileDescriptor** | Descriptor that can be passed between processes |
| **FileDescriptor** | Descriptor that can be passed between processes |
| **IOException** | Deals with communication exceptions |
| **FileInputStream** | Reads from USB |
| **FileOutputStream** | Writes to USB |
| **PendingIntent** | Intent that can be passed to and run by another application |
| **IntentFilter** | Filters Intents for a BroadcastReceiver |
| **BroadcastReceiver** | Listens to discover when a USB accessory has been attached |

**Table 12: Additional modules used in a USB-based sensing application written without using ODK Sensors.**

#### *3.5.3.3 Sensor Driver Developer*

The framework shields device driver developers from tasks required for the app to interact with multiple sensors, such as management of channels, threads, buffers, and sensor states. The device drivers in the framework are designed to be stateless processors of data, which helps developers keep their code simple. Slightly more experienced developers will need to implement sensor drivers because they must be able to interpret sensor data sheets to determine how to manage sensor-specific communications, configuration parameters, and data formats. Table 10 lists the number of lines of code needed to create drivers for the application-specific sensors discussed in Section 3.4.

To simplify driver implementation, driver developers can use a base class provided by the framework development team to handle the communication between the driver and the framework. Since the communication version information is contained within the provided base class, the framework will automatically communicate via the correct protocol when discovering the driver, thereby enabling drivers with varying versions of communication base classes to exist simultaneously on the same Android device. This eliminates the requirement of upgrading all driver applications to the newer protocol version simultaneously with updating the framework. Additionally, upgrading the driver to the latest framework communication protocol should be as simple as swapping in the latest binary containing the base class (assuming no changes to the driver interface functions).

Driver developers also benefit from the framework's system design of deploying drivers through an Android marketplace. When device manufacturers update their drivers, they simply need to post the updated application to market; the built-in Android application update system will then push out the update to devices that previously installed the driver application.

## 3.6 Discussion and Future Work

The long-term goal of the ODK Sensors framework is to enable a market of reusable application components and drivers that can be easily integrated by non-technical users to create sensor-based applications. Deploying mobile sensing applications in under-resourced environments is also challenging because in many circumstances the end-user will likely need to load the sensing software onto consumer devices that may be locked by their service providers. Therefore, for ODK Sensors to be easily deployable by end-users with limited technical abilities, the framework and user-level drivers must fit in to Android's app distribution model.

The concept of user-level drivers is well studied and known to have many advantages, such as ease of development, portability, and maintainability, but generally suffers in terms of performance. The experiments in Section 3.5.1 show that the performance of all three frameworks is adequate compared to overall system throughput (which is limited by the communication channels Bluetooth and USB). Since framework performance does not significantly affect the system, other factors that lower programming and deployment barriers quickly become more important when choosing the optimal framework architecture. One advantage of the V2 and V3 designs is the separation of sensor drivers from the framework, which provides a sandbox environment to minimize negative effects of misbehaving third-party driver code on the framework. In this respect, V2 and V3 are better framework choices because sensor drivers are isolated as separate Android apps and run in their own virtual machines.

A disadvantage of V2's design is that its IPC mechanism acts as a blocking call, which can be potentially dangerous if the driver causes the framework to block forever. However, since each call is isolated in its own thread, the effect of misbehaving drivers is minimal: the framework can handle such drivers with timeouts or exceptions. The main disadvantage of the V3 design is that it is inherently less secure than V2. Since any program can register to receive broadcasts, it is easy for rogue programs to eavesdrop or inject data. This security problem could be addressed by encrypting data sent between the different processes; however, V3 has other limitations, such as timing issues due to broadcast messages that cannot be received until after the framework's *onCreate* method completes. This method is called when an application binds to the ODK Sensors framework; however, no communication between drivers and framework is possible until after construction is complete because no messages can yet be received by the framework. Unfortunately, once framework construction completes, applications can send messages to sensors before the framework's driver connections are properly established, causing timing issues. Based on these implications, we chose V2 as the ideal framework to use for driver development.

With sensors becoming increasingly popular on mobile devices, the possibility of multiple applications simultaneously leveraging the same sensor arises and leads to contention for sensor control, i.e., when different applications using the same sensor issue conflicting start and stop commands. Additionally, it becomes more difficult for the sensor framework to know how long to cache a copy of the sensed data so that it can deliver the data to all waiting

applications. For example, if two applications simultaneously poll a temperature sensor at different rates, the framework must maintain data until both applications receive their copy. However, this can be challenging if one application decides it no longer wants the data and sends a stop command while the other application is still waiting. Techniques are needed to resolve conflicts in resource sharing, leader selection, resource ownership, etc. Currently, the framework avoids these issues by allowing only one application at a time to own an external sensor.

In the future, we plan to explore models for electing a 'leader' application based on the application that is most dependent on the current stream of data and would be most adversely affected by changes to the sensor. A WiFi channel manager is currently under development, and, moving forwards, we will implement an NFC channel manager to manage access to the new class of NFC-enabled, low-power sensors that are becoming available.

## 3.7 Related Work

This section compares ODK Sensors to sensing frameworks developed by others for research or commercial purposes.

### 3.7.1 Amarino

Amarino [46] is a toolkit for the rapid prototyping of mobile, ubiquitous computing applications that connects Android phones to Arduino microcontrollers over Bluetooth. It aims to simplify the development of tangible interfaces that can be controlled by mobile devices. Amarino includes an Android app that eliminates the need to do any Android programming for the kinds of applications it targets and provides an Arduino library that reduces the programming needed to interface with Android over Bluetooth. The Android app lets users connect to an Arduino and select from a set of built-in phone events (e.g., sensor events, phone call or SMS events, etc.) that could be sent over to the Arduino. Developers can also create Android apps that send or receive custom events to/from the Amarino Android app. The Ardunio library lets developers receive and process data from Android and send data to Android (e.g., data from sensors connected to Arduino).

Like ODK Sensors, Amarino lowers programming barriers but only for a certain class of mobile applications (i.e., tangible interfaces built using Arduino). In contrast, the ODK Sensors framework focuses on turning an Android device into a platform that can connect to a wide variety of sensors and actuators over different communication channels to support different

classes of applications (Section 3.4). Arduino can be considered analogous to the sensor-interfacing Bridges in the ODK Sensors system. However, Bridges in our system are not restricted to a specific communication channel, or even to the underlying hardware platform of the Bridge. Additionally, sensors that already have a high-level communication interface (e.g., Bluetooth, USB) can be interfaced directly to ODK Sensors over the appropriate communication channel. For instance, the mPneumonia application (cf. Section 7.4) leverages a pulse oximeter that connects directly to the Android over USB.

### 3.7.2 Reflex

Reflex [47] is a suite of runtime and compilation techniques that conceal the heterogeneous, distributed nature of the underlying system and reduce power consumption by offloading sensor data processing to lower-power co-processors (much like LittleRock in Section 2.4.2.2). It simplifies the development of sensing applications on heterogeneous mobile platforms such as the TI OMAP4 [48] and focuses on improving the energy efficiency and performance of such applications that interact with built-in sensors. ODK Sensors focuses on lowering programming barriers for application developers and supporting different data and application types that interact with either built-in sensors or external sensors. In fact, it could leverage systems like Reflex to improve its energy efficiency while addressing its goals. Since Reflex focuses on sensors built into mobile devices, extending the sensing capabilities of such devices via external sensors is not a prime concern for that system. Thus, simplifying the installation and maintenance of sensing applications via downloadable sensor drivers (from app marketplaces) is not as relevant to Reflex.

### 3.7.3 Gadgeteer

Gadgeteer [49] is a rapid prototyping platform that eases development with embedded hardware devices through the use of modular hardware components and object-oriented programming in C# or Visual Basic. Gadgeteer and ODK Sensors both focus on making it easier for developers to interface with external devices. Gadgeteer achieves this by simplifying how different hardware pieces communicate, e.g., by standardizing connection interfaces and providing a Visual Studio SDK that simplifies how the hardware components are pieced together in software to build a functional system. In contrast, ODK Sensors aims to make it easy for mobile application developers to leverage a variety of sensors in their applications without

significant programming knowledge about the sensors or even about sensor interfacing platforms. Additionally, Gadgeteer does not have a concept similar to our system's sensor drivers, which shield application developers from having to process sensor data received in sensor-specific formats. However, Gadgeteer could be used to quickly build a functional physical computing system and act as an interfacing board for ODK Sensors.

### 3.7.4 PRISM

PRISM [50] is a sensing framework for Windows Mobile that provides reusable components to handle distributed operation, security and privacy. PRISM has been evaluated for a variety of applications, all of which interface exclusively with sensors built into the phone. It also focuses on deploying large-scale remote sensing applications that rely on a backend hosted in the cloud. In contrast, our framework focuses on simplifying interactions with external sensors (in addition to those built into the device) by abstracting away the communications and sensor-specific data processing components. The applications developed using ODK Sensors may or may not have a cloud component. Like Reflex, simplifying the installation and maintenance of sensing spplications via downloadable sensor drivers is not as relevant to PRISM due to its built-in sensor focus.

### 3.7.5 Phidgets

Phidgets [51] provide a simple, "plug and play" method to build sensing and actuation systems that communicate over USB. On the hardware side, they offer USB sensing hubs as well as a variety of USB sensors. On the software side, they provide multi-platform APIs to communicate with their hardware. The APIs are also supported on some mobile platforms, including Android, iOS and Windows Mobile. A Phidgets-based system is similar to ODK Sensors with its USB Bridge. However, it requires Android devices to have USB Host capability, which limits the devices on which it can be used.

## 3.8 Conclusion

The platform shift from traditional PCs to mobile devices with cloud services creates a need and opportunity to integrate these devices (*e.g.,* smartphones, tablets) with external sensors and deploy applications in new settings. To address this, we created an application-level driver framework that enables convenient reuse of sensor-specific code between applications by logically separating the high-level application from the underlying sensor driver. ODK Sensors

enables the integration of data from a variety of sensors over both wired and wireless communication channels. It simplifies application development by creating a single interface that can control any kind of sensor (either built-in or external) and reduces the amount of code needed to access a sensor. Applications that leverage the framework to interface with external sensors can be implemented in fewer lines of code (Table 10) because ODK Sensors handles the communication on behalf of applications. Additionally, the sensor framework automatically multiplexes the communication channels, allowing different types of sensors to be used simultaneously by an application. For example, an application can easily use two USB and three Bluetooth sensors simultaneously to record several phenomena at once. The ODK Sensors framework is designed to flexibly meet any application's needs regardless of data type, data collection rate and size, sensor configuration requirements, or communication channel.

We evaluated three framework implementations, after which it became clear that performance was not the most important factor to consider when selecting the final design since most sensing applications sample data at a lower rate than the framework's maximum throughput. Our performance analysis showed that the system bottleneck is the throughput of the Bluetooth and USB communication channels, not the framework throughput. Since the three frameworks performed similarly, other factors were examined before deciding which was optimal. The V2 framework offered the best trade-off in terms of programming ease, deployment ease, and performance.

ODK Sensors can be used in conjunction with the interfacing boards described in chapter 2 to enable a wide range of applications, some of which are presented in Chapters 4 through 7.

# Chapter 4

# Vaccine Cold Chain Monitoring

This chapter presents a mobile phone-based, real-time sensing system to remotely monitor temperatures in a vaccine supply chain, which leverages the building blocks presented in Chapters 2 and 3. Our vaccine monitoring work was done in collaboration with PATH [20], a Seattle-based non-profit organization, and the World Health Organization. While it focuses on a specific application, the work can be generalized to other domains, such as blood or dairy supply chain monitoring and large-scale environmental monitoring. This chapter provides evidence for contribution 3 of the thesis: **Implementation and evaluation of sensing systems to enhance workflows in low-resource settings.**

We begin by describing the need for real-time temperature monitoring in vaccine supply chains. Section 4.2 reviews our system, while Section 4.3 discusses lab experiments we performed to verify the functionality of our system. Section 4.4 describes a field trial of the system done in Albania in 2010. Lessons learned from that trial led to a new implementation, discussed in Section 4.5. We position our work in the context of others' related work in Section 4.6 before concluding the chapter in Section 4.7.

## 4.1 Motivation

Effective management and delivery of vaccines is an important aspect of improving healthcare in developing countries. The World Health Organization (WHO) estimates that vaccines prevent over 2.5 million child deaths from preventable diseases annually [52]. Access and availability of vaccines has improved tremendously over the last five to ten years with the influx of donor resources and increased supplies from several new manufacturers. However, delivery and storage of these temperature and time sensitive vaccines continues to be challenging in many developing countries. WHO specifies the procedures to be followed for storing vaccines safely, which include routine temperature monitoring and reporting of alarm conditions (e.g.,

storage temperatures deviating from the allowable range for a prolonged period of time) [53], [54]. Vaccines must be stored in a temperature-controlled environment (+2°C to +8°C for most vaccines) until they are administered to patients [54]. The temperature-controlled supply chain used for distributing vaccines is known as the "cold chain". Figure 21 shows a few pictures of equipment used to store and transport vaccines in a cold chain.

**Figure 21: Refrigerators and cold boxes used for vaccine storage and transportation. Photo credit: Prof. Richard Anderson, UW, Seattle, USA.**

Refrigeration requirements for vaccines are more stringent than for many foods and pharmaceuticals since they contain biological agents that are permanently damaged if frozen. However, country cold chains are not always reliable; as a result, vaccines may be exposed to temperatures that are either too hot or too cold, causing vaccines to spoil. Studies have shown that vaccine spoilage due to freezing (caused by malfunctioning equipment) is a more frequent (and serious) problem in cold chains than damage due to exposure to high temperatures. ([55]–[58]). Intermittent power outages, equipment malfunction, limited availability of skilled staff for equipment maintenance, lack of spare parts, and interruptions during transportation are common causes for such temperature excursions [59]. The high cost of vaccines, coupled with the fact that even short durations of exposure to temperatures outside the normal range can cause irreversible damage, necessitate the need for robust storage equipment and continuous temperature

monitoring and reporting systems. The vaccine market segment in developing countries is estimated to be about $4 billion USD [60], of which 5% is spoiled annually [61]. More importantly, spoiled vaccines lose their potency and, if administered, do not provide protection against diseases.

A countrywide vaccine cold chain is a hierarchical system with national-level storage facilities that distribute vaccines to state/district-level storage facilities. These regional facilities then distribute to local healthcare facilities that administer vaccines to patients. Temperature monitoring is critical at all levels of the system. Further, there are several stakeholders in this system who have different roles and information needs. The supervisory staff at storage facilities ensures that the supply chain in their region works efficiently; this includes logistics management and monitoring the operational efficiency of cold chain equipment. Store managers at these facilities perform day-to-day operations, which includes distributing vaccines to sub-regional levels, monitoring the temperature of storage equipment, and troubleshooting problems such as power outages, equipment failures, etc. Finally, staff at healthcare facilities administer vaccines to patients and ensure that vaccines are stored safely.

Cold chain temperature monitoring devices and reporting systems must be simple, low-cost and scalable due to the countrywide scale of operation. A variety of monitoring devices are used for recording temperature. Thermometers provide the simplest and cheapest way to monitor temperature; however, they lack some important features needed for cold chain monitoring. For instance, they show only the current temperature; information such as the daily maximum/minimum temperatures, deviations from the normal temperature range, etc., is not available. Products at the next level up in price and complexity provide these missing pieces of information, including persistent history over a period of time. These cost-effective devices are widely used in cold chain management. However, they lack communication interfaces to transfer temperature data from the device to a central database. Commonly used procedures require facility staff to manually record temperature twice daily on paper (per WHO guidelines [54], page 40). This method can produce accurate results (depending on the accuracy of the thermometer used and compliance of the staff performing the procedure). However, manual recording on paper makes data aggregation difficult, especially when facilities are not geographically co-located. Figure 22 shows the paper-based method used at a health clinic in Nicaragua to record the temperature of equipment used for vaccine storage.


**Figure 22: Paper-based temperature reporting at a health clinic in Nicaragua. Photo credit: Prof. Richard Anderson, UW, Seattle, USA.**

The capability to easily move data off the monitoring device has potential to significantly improve cold chain management. For instance, alarm notifications can inform staff about problems in real-time, and detailed temperature profiles accessible from a central server can help identify poorly performing equipment. Remote monitoring can provide data to several stakeholders simultaneously, enabling better supervision of operations. While more expensive alternatives provide many of these capabilities, their high cost, coupled with installation and management complexities, prevents scale up to provide countrywide cold chain coverage in many developing world contexts.

Evolution of vaccine technology might make it possible to safely store vaccines over a wider temperature range [62]. For instance, future vaccines maybe storable at room temperature. However, monitoring systems to ensure that vaccines remain within the acceptable temperature range would still be required. This fact must be considered as we build monitoring systems today: our systems must be flexible and reconfigurable to easily adapt to changing requirements for vaccine storage.

## 4.2 System Overview

Our remote temperature monitoring system is based on FoneAstra 1.0 and low-tier mobile phones (discussed in Chapter 2). It occupies a middle ground between existing solutions because it provides the capabilities of more expensive systems at a low-cost while using familiar

components (low-tier mobile phones) and communication methods (SMS). It does not require Internet access, which is difficult in environments with transitory power supplies and only basic cellular service and phones. It is cost-effective to use low-tier mobile phones, which cost about $20 USD, as communication modems as opposed to cellular modem modules (e.g., [63]), which can cost up to $60 USD. Moreover, the phone's user interface is very helpful in troubleshooting system problems in the field. Specific examples include: low battery indicators, signal strength indicators, the ability to test SMS sending and receiving capability, etc. Based on our interactions with deployment partners in developing countries, we have come to realize that the practical benefits of using mobile phones significantly outweigh even the cost benefits.

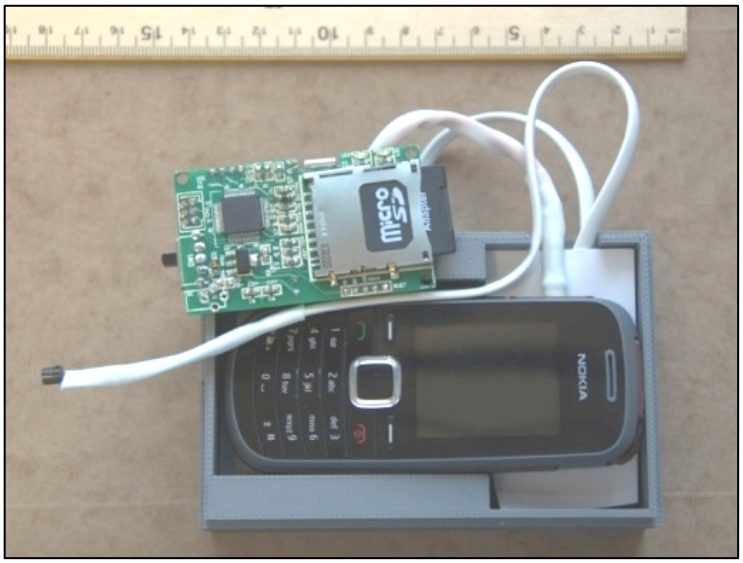



**Figure 23**: **FoneAstra with a waterproofed temperature sensor (at the end of the white cable used to place it appropriately within a cold box) and connected to a Nokia 1661 mobile phone. The SD card is visible on the right side of the FoneAstra board. The ensemble is packaged in the gray box visible around the phone (the final dimensions are 12cm X 8cm X 5cm).**

Our monitoring system combines a FoneAstra board, a connected temperature sensor, and a mobile phone, as shown in Figure 23. While the figure shows only one temperature sensor connected to the board, several sensors can be connected simultaneously. (We have tested up to ten sensors simultaneously connected to the board, but in theory this number can be much higher.) We use the DS18B20 [64] temperature sensors, which implement the 1-wire protocol [65] to interface with a host; this allows multiple 1-wire devices to be connected to one pin of the board's processor. FoneAstra can be powered with its own battery or share the phone's battery. The phone and FoneAstra can run for up to two weeks (depending on the rate of SMS messages and the number of sensors connected) on a typical fully charged phone battery. We assume the

availability of sufficient power to keep the battery charged; this is the case when placing FoneAstras near refrigerators. The total cost of the ensemble shown in Figure 23 is about $50 USD.

For this application, the software elements of FoneAstra are built on top of FreeRTOS [19], an open-source, general-purpose embedded real-time operating system. The system contains a set of cooperative tasks that execute concurrently in the FreeRTOS runtime environment. Much of the application-specific logic is built into a task that periodically samples the on-board sensors. Another task listens for, and processes, incoming SMS messages, while a third task controls the on-board LEDs.

The server component of the FoneAstra system is based on RapidSMS [66], an open-source SMS gateway, and Django [67], an open-source web application framework.

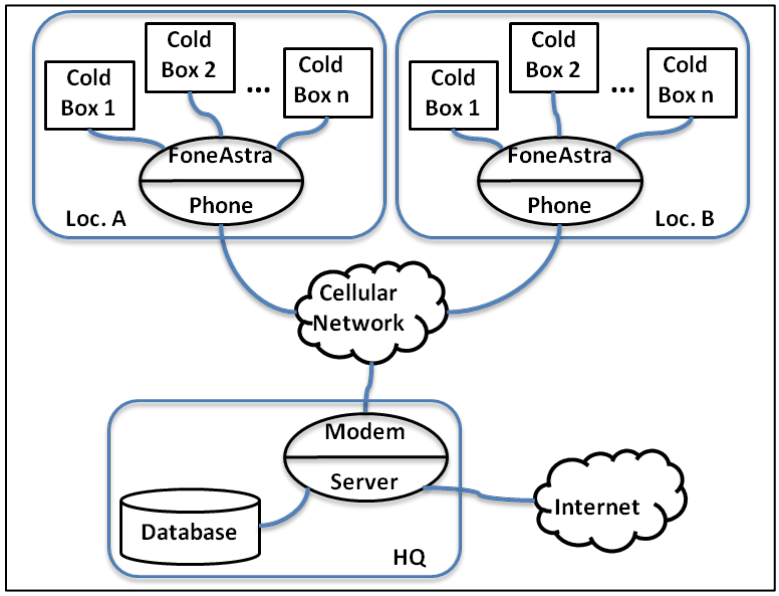



**Figure 24: End-to-end architecture of the FoneAstra cold chain monitoring system. Phones coupled with FoneAstra are deployed at vaccine storage facilities. Each FoneAstra can monitor several temperature sensors simultaneously. The phones communicate to a central server using SMS messages over the cellular network. The server receives the messages through its cellular modem, stores the data in a database, and permits users to browse the data through an Internet browser.**

Figure 24 shows a schematic for a typical deployment of FoneAstras. Each location has a FoneAstra board and phone and a multitude of sensors. More than one device can be placed at a single facility if required. All devices communicate through SMS messages over the cellular network to the headquarters, where a server backs up data into a master database. A web

interface provides visualization of this data through standard web protocols. Typically, the FoneAstras are programmed to send a periodic summary message as well as to send alarm messages if particular conditions are detected.

The server can send commands to the FoneAstras to adjust sampling rates, clear storage, change alert thresholds, or request more data in addition to the summaries regularly sent. This level of control may be necessary to deal with dynamic situations and/or control SMS costs for a particular deployment. For example, it could be advantageous for the FoneAstras to send alarm messages only, while someone visiting each site weekly or monthly could read the SD card to collect the logged data.

Figure 25 shows the SMS format for messages sent *from* FoneAstra and the details of a typical SMS report message from a facility where two refrigeration units were being monitored. Messages sent *to* FoneAstra are much simpler; for example, an SMS containing "@FAQ" queries FoneAstra for current sensor values and return these values to the sender. This is made purposely short and simple so a human can send it easily from his or her own phone and receive a direct reply. Further details of the system, including alternate architectural instantiations and power consumption are discussed in Chapter 2.

| | Header | | | | Data Payload Per Sensor | | | |
|---|---|---|---|---|---|---|---|---|
| Size (chars) | 1 | 4 | 1 | 2 | 2 | 2 | 2 per data sample | |
| Field Name | Type | Seq. no | Retry count | No. of sensors | Sensor-id | No. of samples | | |

**Typical message from FoneAstra:**

```
1 0069 1 02
00 30
1V 1Z 1d 1h 1k 1o 1q 14 1D 1M
18 0u 1L 1G 11 0x 1I 1D 0y 1E
17 15 12 10 0/ 0z 0z 0x 0x 0v
01 30
1R 1R 1d 1G 1b 18 1Y 1M 1T 1e
1L 1d 19 1a 1D 1W 1O 1S 1e 1K
1c 1C 1b 19 1X 1N 1S 1e 1J 1c
```

Header specifies there are 2 sensors

ID for sensor 00 followed by 30 samples

ID for sensor 01 followed by 30 samples

**Figure 25: Format of SMS messages sent from FoneAstra (above) with an illustrative example (below, with extra spaces for readability). Sensor values use 2 characters – the value is encoded in 6 bits of each of the 7-bit SMS ASCII characters as a 12-bit 2s-complement value.**

## 4.3 In-Lab Experiments

Before trialing the system in a developing country cold chain, we set up an experimental test bed at PATH's lab in Seattle. A temperature probe connected to FoneAstra monitored the internal temperature of a solar powered cold box, shown in Figure 26. We configured the sampling interval on FoneAstra to be higher (30 seconds to 2 minutes) than required by a real cold chain monitoring system. Additionally, routine reports were generated every hour to provide us with detailed measurements rather than being generated once every several hours, as is the norm for a typical system deployed in the field. The experiments we conducted and results obtained are discussed in this section.


**Figure 26: A solar powered cold box enhanced with temperature monitoring. FoneAstra and the connected Nokia 1661 mobile phone are placed on top of the cold box. The temperature sensor connected to FoneAstra is placed inside the cold box. Photo credit: PATH, Seattle, USA.**

### 4.3.1 Ground Truth Validation

The datasheet of the DS18B20 (the 1-wire temperature sensor we use) specifies that the sensor is accurate to within 0.5$^{o}$C in the –10$^{o}$C to +85$^{o}$C temperature range; however, the sensor is not waterproof as it ships. Since we were using the sensor in environments where moisture content is high, we waterproofed the sensor by insulating its body and connecting leads with brush-on electrical tape. (Figure 23 shows the sensor after it had been waterproofed.) To ensure that the waterproofed sensor performed at its specified accuracy, we compared the sensor's

temperature readings with the readings from a type-K thermocouple connected to a Fluke multimeter [68]. The graph in Figure 27 shows the results of our ground-truth validation experiments. For this experiment, the thermocouple and the DS18B20 were taped to the same location inside the cold box. FoneAstra was configured to sample the sensor every 30 seconds. The internal temperature of the cold box was +24.5$^{o}$C (room temperature) when the experiment was started.

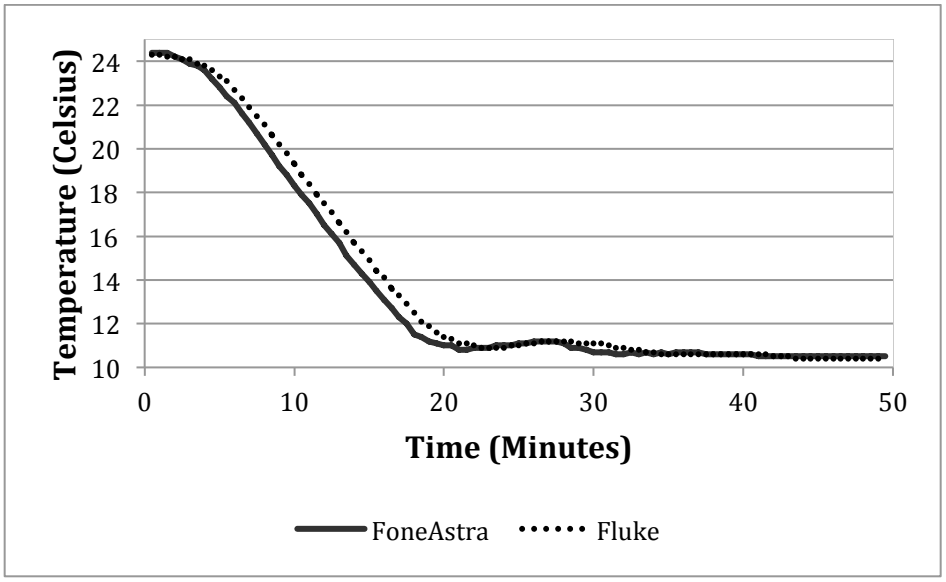



**Figure 27: Ground truth validation of FoneAstra's 1-wire temperature sensor. In this experiment, FoneAstra's temperature sensor and a type-K thermocouple were placed inside the cold box for about an hour, and box's lid was closed. Initially, the internal temperature of the cold box was high, close to room temperature. Over time, the temperature dropped and stabilized at around +10.0$^{o}$C. The curves from the two sensors are highly correlated, indicating that FoneAstra's sensor was close to the thermocouple in accuracy and responsiveness.**

Temperature readings from the Fluke thermocouple were manually recorded every time FoneAstra queried its temperature sensor (it was programmed to blink an LED when it sampled the temperature sensor). We ran this experiment for about an hour (the graph shows readings from the first 50 minutes) until the internal temperature of the cold box stabilized at approximately +10.0$^{o}$C. As can be seen from the graph, the temperatures reported by FoneAstra and the Fluke thermocouple were very close throughout the experiment. FoneAstra reported temperatures that were either slightly lower than or equal to Fluke's readings, but the readings from the two devices eventually converged to +10.4$^{o}$C when the internal temperature of the cold box stabilized. The minor difference in readings could be caused by temperature sensors being

sampled at different rates, and the exact time at which each sensor was sampled most likely differed for each device. Based on this experiment, we concluded that waterproofing had not modified the thermal characteristics of our temperature sensor.

### 4.3.2 Detecting Power Failure

During our in-lab testing, the power supply to the cold box was inadvertently disconnected for a few hours. This provided us with data simulating a power failure in the field. Before the power was disconnected, the cold box had been running continuously for about a week, and FoneAstra had been monitoring the temperature during this time. FoneAstra was configured to sample its temperature sensor every 2 minutes and report readings every hour. Figure 28 shows an almost flat temperature curve until Time=132 minutes. This had been the stable temperature over the weeklong continuous operation. The power supply was disconnected at Time=132 minutes, and, as the graph shows, the temperature started rising rapidly thereafter. Power was resumed at Time=442 minutes, and the cold box returned to its stable temperature shortly thereafter. This experiment demonstrated that it was possible to determine power failures at facilities based on cold chain equipment's temperature profile. This capability can significantly aid in reducing vaccine spoilage due to power failures in cold chains.

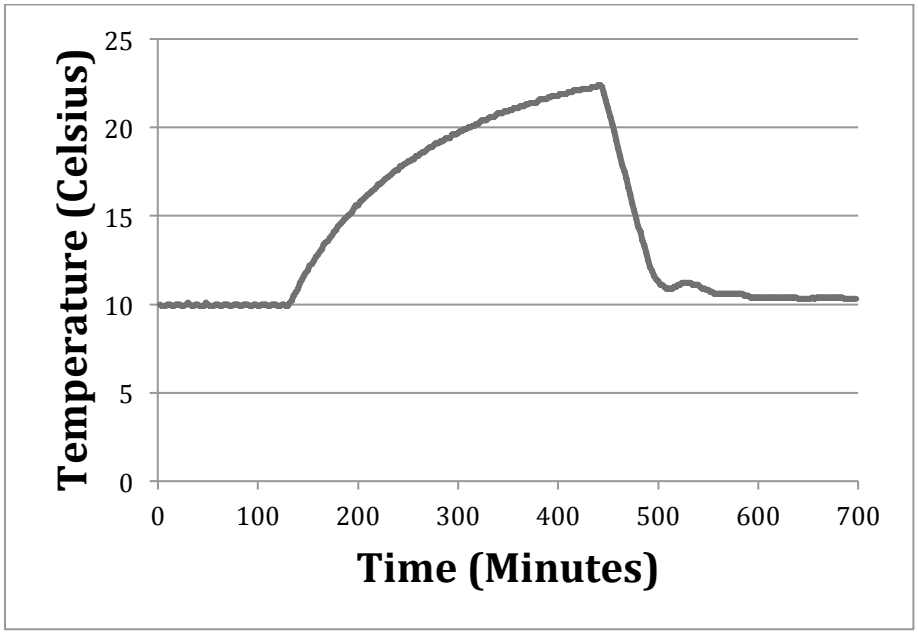



**Figure 28: Temperature curve of the cold box when its power was interrupted for a few hours. Power was disconnected at Time = 132 minutes, and the temperature then started rising. Power resumed at Time = 442 minutes, and the cold box returned to its stable temperature shortly thereafter.**

### 4.3.3 Detecting Lid Open/Close Events

Figure 29 shows a graph from another experiment that configured FoneAstra to sample the sensor every 30 seconds. In this experiment, we opened the lid of the cold box after it had reached its stable operating temperature, approximately +10.4$^{o}$C, and left it open for about 30 minutes. As expected, the temperature started rising shortly after the lid was opened. The temperature curve was steeper when the lid was left open than when the power to the cold box was disconnected (shown in Figure 28): the cold box's internal temperature increased rapidly when the lid was left open due to the large difference between the internal temperature and room temperature. The lid was closed at Time=28 minutes, and the temperature started to drop at that time. The temperature stabilized again around +10.4$^{o}$C, at Time=50 minutes. This result indicated that the equipment's temperature curve could also be used to detect if its lid (or door) were left open by mistake, a scenario that can be differentiated from a power failure. Additionally, we hypothesized that the operational efficiency of equipment could be inferred based on the time taken for internal temperature to stabilize after the lid is closed.

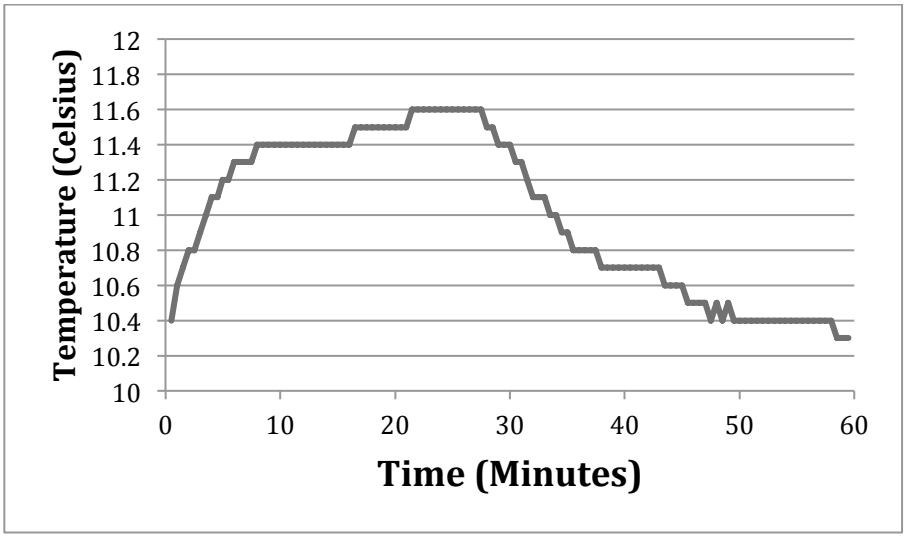



**Figure 29: An experiment showing the possibility of detecting cold box lid open and close events. This plot begins when the lid was opened, and it was left open for about 30 minutes. As expected, the cold box's temperature started rising. However, the temperature rises more rapidly compared to the power failure scenario (Figure 28) due to the large difference between the cold box's internal temperature and the external room temperature. The lid was closed again at Time = 28 minutes. This experiment indicated that the temperature curve could be used to detect if the lid were left open by mistake, differentiating the increase in temperature from a power failure event.**

We performed additional experiments in which we opened the lid for short durations of time (30 seconds to 2 minutes), simulating normal usage scenarios for the cold box. However,

instead of shutting the lid completely, we left it ajar for an extended period of time, thus simulating scenarios where the cold box lid was not closed properly. As expected, we saw the temperature rise (by 0.2°C to 0.5°C) when the lid was opened, and it remained high for a short duration as the lid was left ajar. Interestingly, the cold box recovered from this anomaly quickly, and its internal temperature returned to its stable value. With the lid ajar, the cold box was losing energy while maintaining its internal temperature, so its current drain increased slightly (0.1 to 0.2 amps); this was also as expected. However, when the lid was closed, we noticed that the current drain did not drop. We monitored the current for about 30 minutes after closing the lid and did not see current consumption decrease. We were unable to explain this behavior but hypothesized that it might indicate a problem. If a refrigeration expert could validate this hypothesis, and if this were a commonly occurring equipment problem, it might be useful to include a current sensor in cold chain monitoring devices to monitor changes in current. This could help with timely detection of equipment problems.

## 4.4 Deployment in Albania

In collaboration with PATH, WHO and the Albanian Ministry of Health, we trialed the FoneAstra-based continuous monitoring system in Albania in November 2010. The Albanian national-level vaccine storage facility is located in Tirana (the capital city), at the Institute of Public Health (IPH). The IPH has four cold rooms where vaccines are stored until they are transported to district-level storage facilities. The IPH supplies vaccines to 36 district-level stores once every four months. The district stores supply vaccines to their local health centers and health posts every month. While the exact number of healthcare facilities under each district is not fixed, Albania has a total of about 500 health centers and 1500 health posts across the country. The district stores have multiple large refrigerators for storing vaccines, while the health centers and health posts typically have one standard refrigerator per facility.

We deployed six FoneAstras to monitor equipment at five different facilities. Our deployment covered the four cold rooms at the IPH, two refrigerators at Tirana's district-level store (called the Directorate of Public Health, or DPH), and individual refrigerators at three health centers in the district. The cold rooms at the IPH are grouped into two pairs separated by an aisle. Two FoneAstras, with two temperature probes connected onto each device, monitored these rooms. Figure 30 shows one pair of cold rooms with the compressors for each room mounted on the wall next to their respective doors. A FoneAstra was mounted on a panel in the

space between the two compressors. Temperature probes from each device were routed into the cold room through thermally insulated holes in the rooms' walls.


**Figure 30: FoneAstra deployed at the country's national store in Tirana, Albania. The FoneAstra box was placed between the two compressors for two cold rooms (note each room's door handles on left and right). Cables from FoneAstra were routed through thermally insulated holes in the walls of the cold rooms and lead to the sensor inside. Photo credit: Ministry of Health, Albania.**

The two refrigerators monitored at the DPH were located next to each other in a room; therefore only one FoneAstra was deployed at this facility. Since only one refrigerator was monitored at each of the health centers, one FoneAstra with one temperature probe was deployed at these facilities. Figure 31 shows a FoneAstra placed on top of a refrigerator at a health center, while the inset is an interior view of the fridge and shows the tip of FoneAstra's temperature probe. A FridgeTag [69] temperature monitor and some vaccines stored in this fridge are also visible in this picture. We used this arrangement to compare the readings from the two devices.

**Figure 31: FoneAstra deployed at a health center in Tirana. Note the box on the top back-right corner of the household fridge and the wire going down to the door and through the insulating lining. The inset shows the temperature sensor hanging inside the fridge, just above a FridgeTag for comparing temperature readings from both devices. Photo credit: Ministry of Health, Albania.**

The FoneAstras were configured to sample the temperature sensor(s) once every four minutes and send an SMS message to the server every two hours. These periodic SMS messages (referred to as routine reports) contained the temperature data aggregated over the two-hour reporting interval. Alarm notifications were sent to the server as soon as an alarm condition was detected. The local staff could remotely query the temperature of monitored equipment at any time by sending an SMS message to a FoneAstra with a predefined command. These configuration parameters (sampling interval and reporting interval) were chosen because of the cellular plan, which cost about 4 Euros per month and included 500 SMS messages in each billing cycle. In this configuration, each FoneAstra generated 360 (or 372) routine reports per month. The remaining monthly quota of messages was used for responding to SMS queries sent to the device and sending alarm notifications.

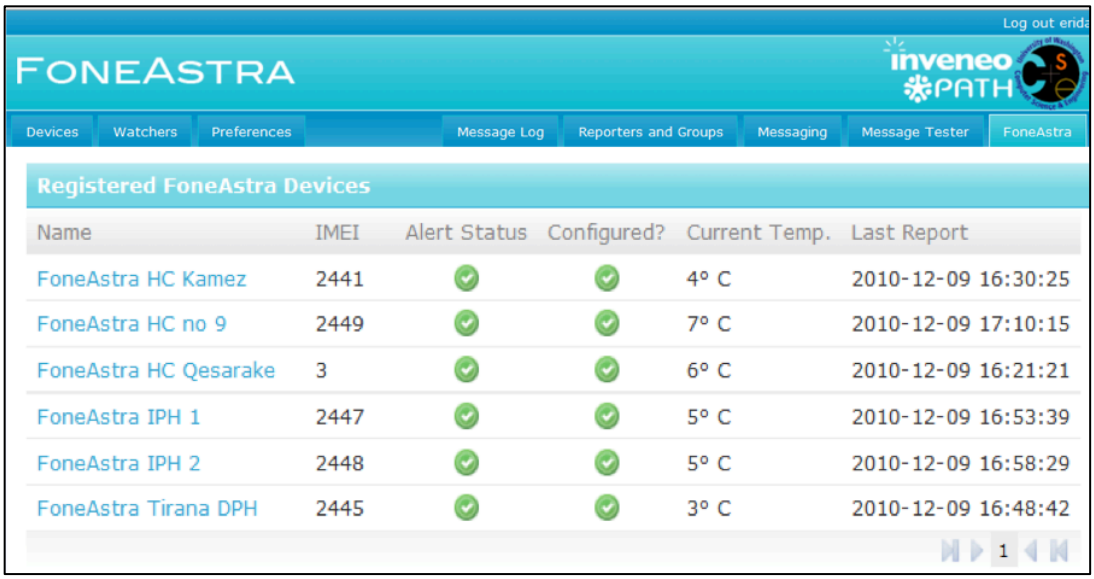



**Figure 32: The server's dashboard showing all the FoneAstra devices in the system and their most recent sensor readings. The alert status icon is green when equipment temperatures are within the normal range and turns red when an alarm event is reported from the FoneAstra at the facility being monitored.**

The server that was shared by all the FoneAstras deployed in the system was hosted in Tirana itself. Users (i.e., cold chain supervisors) logged on to the server via the web interface to access information about the system and configure FoneAstras as needed. Figure 32, a screenshot of the server's dashboard, shows a summary view of all six devices deployed in Albania. Device information displayed on this page includes the timestamp of the last report received and the last known temperature of the equipment being monitored. The alert status column shows whether equipment have an active alarm condition; a green icon indicates normal conditions, while a red icon indicates that an alarm is currently active. Users could view detailed information about a device by clicking on its hyperlink on the dashboard.

Figure 33 is a screenshot of the detailed information of the FoneAstra deployed at the DPH. Displayed information includes the current temperature, the alert status, and configuration parameters like the temperature thresholds for alarm reporting and alert frequencies. Changes to configuration parameters resulted in an SMS message being sent to the device with updated information. Clicking on a sensor hyperlink displays the detailed temperature reported by that sensor. Figure 34 to Figure 37 (discussed in the next section) show screenshots of temperature graphs obtained from a few sensors.

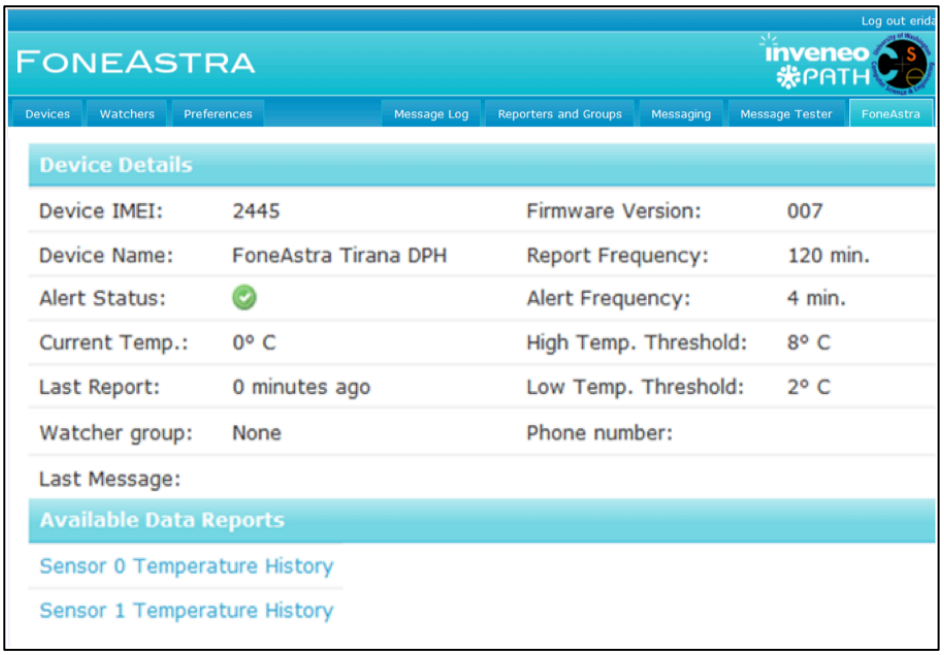



**Figure 33: Detailed view of the FoneAstra deployed at the DPH. Information displayed includes system parameters, like the SMS reporting frequency and temperature thresholds. The FoneAstra at this site had two connected sensors, "Sensor 0" and "Sensor 1" on this web page.**

The system deployment was completed over a 12-day period in November 2010, and it was trialed for 6 months. Some basic training to troubleshoot common system problems was provided to the IPH staff during the initial 12-day period. There were interruptions in operation due to various issues (e.g., a broken connection between FoneAstra and the phone), but the IPH staff were able to resolve them.

### 4.4.1 RESULTS

Results obtained from our Albania trial are discussed in this section.

#### *4.4.1.1 Alarms*

WHO specifies the temperature and time thresholds for various alarm conditions [54]. Being able to precisely and successfully detect alarm conditions in the refrigeration equipment was amongst the early results we obtained from the system.

Within a few hours of deployment at one facility, our system reported negative temperatures (Figure 34). This concerned cold chain supervisors because freezing alarms are potentially more serious than high temperature alarms [55]. This particular equipment was known to have intermittent problems, but its severity was highlighted because the staff got near-real-time feedback from the equipment through FoneAstra and viewed its detailed temperature

profile from the web interface. A software upgrade to FoneAstra on November 11th added support for detecting WHO-specified alarm conditions and sending an immediate SMS notification to the server. The staff now easily saw that the fridge was in fact going into a freezing alarm condition (i.e., temperature was below -0.5° C for at least a 1 hour time interval) at least once every 24 hours. Based on this feedback, vaccines were moved out of this refrigerator while the staff experimented with the refrigerator's thermostat settings to control the situation. Over the next few days, they found a setting that prevented the freezing alarm condition. As shown in the figure, even though the fridge's internal temperature went below 0°C in the time period between November 15th and December 1st, it recovered before getting into the alarm condition. However, the alarm condition was again triggered later in the month. Staff could not find a long-term solution to this problem, and the refrigerator was eventually replaced in early 2011.

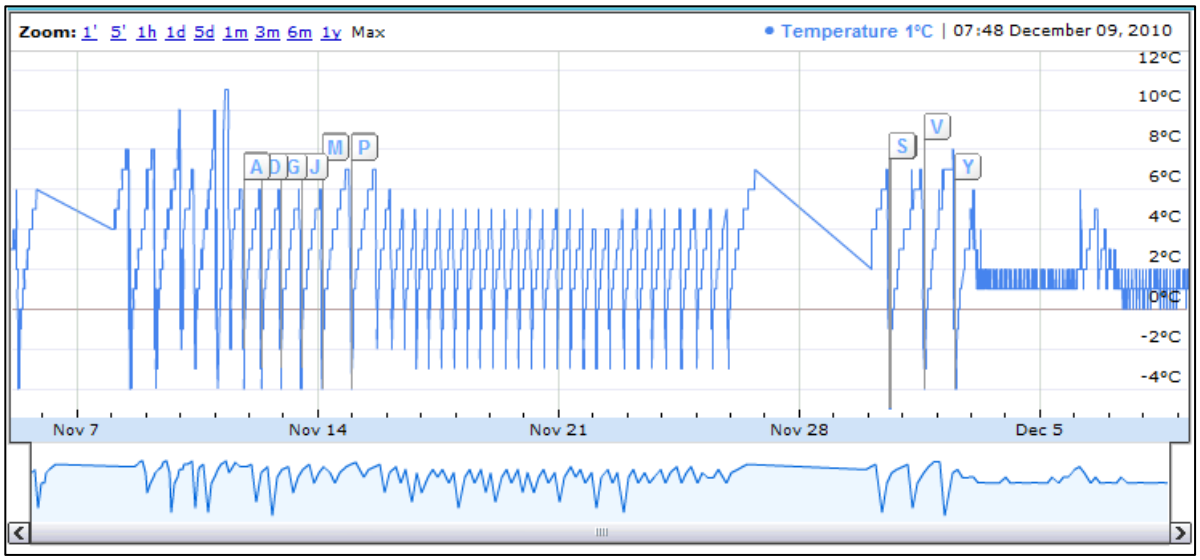



**Figure 34: Freezing alarms (indicated by flag markers on the graph) reported for a fridge. Variations in the temperature profile occurred because facility staff were adjusting fridge settings to resolve the freezing issue identified on this graph.**

Figure 34 also shows two time periods when FoneAstra experienced problems and went offline. These were around the November 7th and November 28th time frame, depicted by the straight line connecting two data points. In the first case, the cable connecting FoneAstra to the mobile phone got disconnected, so the server was not receiving any messages from this device. In the second case, the SIM card had run out of SMS credits. The staff easily diagnosed and resolved both of these issues on their own.

#### 4.4.1.2 Detecting Power Interruptions

Figure 35 shows the temperature graph from a cold room at the national store. The three temperature spikes shown in the graph were due to interruptions in the power supply of the cold room. The huge temperature spike between November 9th and November 11th occurred because the cold room was under maintenance, and its door was left open, in addition to the power interruption. This was actually a WHO high temperature alarm condition (temperature higher than +8°C for at least 10 hours); however, FoneAstra was not configured to generate alarms at the time, and an alarm did not show up on the graph. The other two spikes were due to shorter duration power failures at the facility. The temperature curve for the power failure on November 7th shows a drop in temperature during the power failure: the generator was turned on for a short period of time to bring the room's internal temperature under the high temperature threshold (+8.0°C). This apparently is common practice at the facility and was verified by the country cold chain manager.

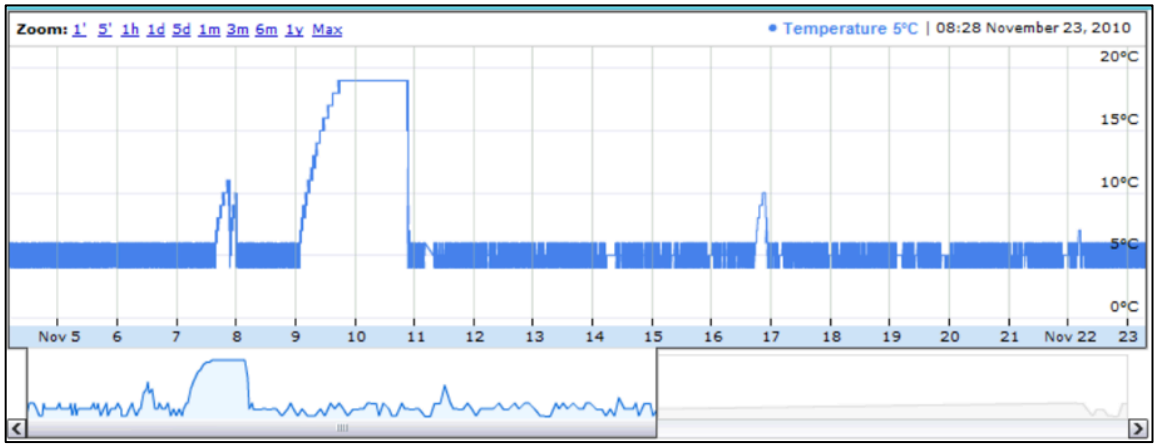



**Figure 35: Temperature graph for a cold room at the IPH being monitored by FoneAstra. The three temperature spikes were due to power interruptions. The temperature drop in the first spike occurred because the generator was turned on for a brief period to control temperature during the power failure.**

#### 4.4.1.3 Performance Profiling

Having a continuous monitoring system gives insights about the operational efficiency of equipment being used in the cold chain. Figure 36 shows the temperature profile of a newly installed fridge that was used as a baseline for performance comparison in the deployment. The inset shows the fridge's temperature curve over a 24-hour period. The uniformity in the temperature curve of this fridge indicates that it was performing efficiently. If equipment generates alarms often, it is obviously a strong indicator of an underlying problem; however, a

lot can be inferred from the temperature profile beyond just the flagged alarms. For instance, the graph in Figure 34 shows periods when the internal temperature was subzero but alarms were not generated by the fridge. This indicates a poorly performing compressor. Figure 37 shows the temperature profile of a fridge that generated a few high temperature alarms due to power failures. The high degree of variation in the temperature profile of this fridge indicates that it was performing poorly. This fridge was also replaced in 2011. While we do not show an enlarged view of the temperature profile of the cold room at the IPH (Figure 35), a quick visual inspection of its graph shows uniformity in the temperature profile (barring power cut-offs), indicating high operational efficiency.

In the future, we expect that we could predict sooner when a fridge is beginning to fail so that it could be serviced before it fails. With an additional sensor, we could easily detect when doors on cold boxes and fridges are opened, and FoneAstra could pay particular attention to how long equipment takes to recover to the preset temperature. Equipment starting to show curves with different characteristics would suggest degrading efficiency and the need for service.

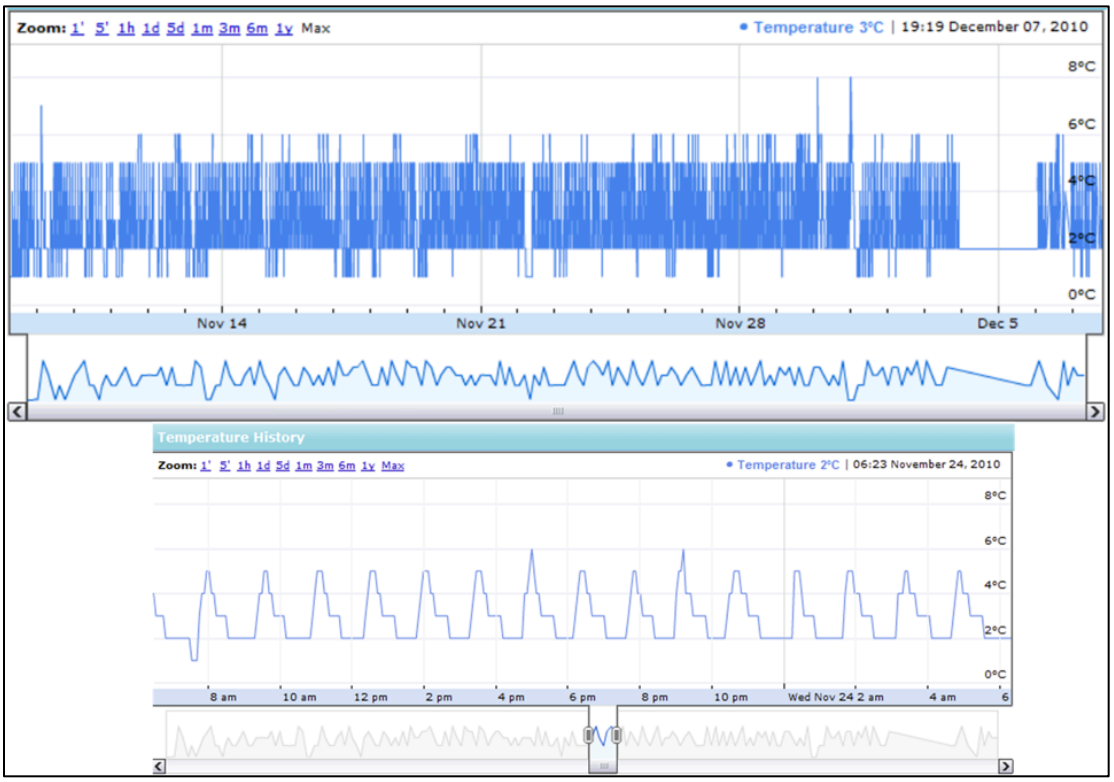



**Figure 36: Temperature graph of a newly installed fridge. The uniformity in the temperature profile indicates high operational efficiency. The inset shows an enlarged view of the temperature profile over a 24-hour period.**

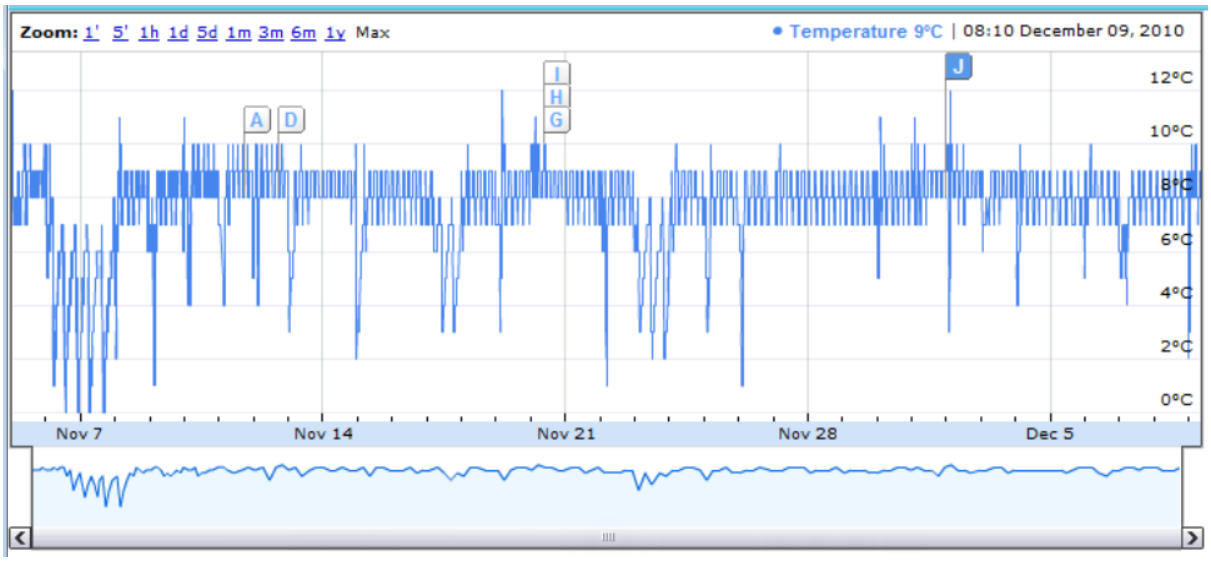



**Figure 37: Temperature profile of a poorly performing fridge. The high degree of variation in the temperature profile indicates poor operational efficiency. Flags on the graph indicate high temperature alarms. At least two of these alarms occurred due to power failures over weekends when the facility was closed.**

#### *4.4.1.4 Summary of Results*

The results from our Albania trial were very encouraging, a fact endorsed by cold chain experts involved in this project. While the local cold chain staff was aware of some of the problems detected by our system, the near-real-time reporting and flexible data visualization gave them a fine-grained, current view into their system. This made them proactive in diagnosing problems and let them intervene in a timely manner to resolve issues. The evidence for this was already strong in the first two weeks of deployment, and it continued through the trial period. Based on feedback from the FoneAstra system, the staff quickly identified poorly performing equipment, and either replaced it or got it serviced.

### 4.4.2 Lessons Learned

Motivated by the positive results from the trial in Albania, we are now developing a more scalable system to provide better coverage of countrywide cold chains. Deploying a smaller system in Albania gave us a better perspective on vaccine cold chains, which is guiding our current efforts.

Our low-cost monitoring solution seemed appropriate for large-scale deployments, but it provides limited local feedback on the device via LEDs. While it is possible to send input commands via SMS, without a DTMF decoder on FoneAstra (cf. Section 2.1.1), it is not possible to send input to the board locally. We have learned that a richer user interface will be needed on

monitoring devices used at facilities that store large volumes of vaccines (e.g., national- or district-level stores). These are critical points in the cold chain, and it is important even for local staff to be able to view temperature profiles of storage equipment. Since the availability of a computer connected to the Internet to access the server cannot be assumed for all facilities, the monitoring device itself needs to be more interactive in order to display temperature profiles locally. Additionally, circumstances often change at all levels of the cold chain. For instance, it is common for equipment to be added or removed from facilities. This suggests that sensors need to be added or removed, and the mapping between equipment and sensors must be updated on both the monitoring device, and the server. Thus, the monitoring device must be more interactive at all levels of the cold chain in order to handle such changes during its lifetime. Instead of enhancing the I/O capabilities of FoneAstra, we are now building a new temperature monitoring system that leverages cheap Android phones; this work is discussed in the next section.

While the value-add of real-time, continuous temperature monitoring was clear in the Albania trial, we realized that it might not be practical to implement at lower levels of the cold chain (e.g., at remote health posts). Even if monitoring devices were cheap, or made available at subsidized rates, the recurring cost of cellular services would pose a barrier that would prevent deployment at such facilities. Additionally, some parts of the cold chain do not have cellular coverage (e.g., remote facilities or air transit routes); real-time monitoring would not be possible in these parts even if funds were available to pay for cellular service. However, both the cold chain experts on this project and the Albanian cold chain supervisors said that getting detailed temperature data from such sites (or travel routes) would be valuable even if it were not available in real-time. Historical data would indicate the operational efficiency of equipment, letting supervisors be proactive in equipment maintenance. Currently available low-cost temperature monitors do not address this requirement appropriately since it is difficult to retrieve historical data from them (cf. Section 4.6). Hence, we are addressing this requirement as well in our new implementation, discussed in the next section.

Administering and maintaining the server in Tirana was inconvenient. The cold chain staff could not do this because they were not IT administrators. At times, the server machine went offline, and it became difficult even for us to access it remotely for troubleshooting. It was clear that a better solution was needed. The server was hosted in-country because it acted as an SMS gateway in addition to being a web server. We have now decoupled these components to

lower the administrative barrier. Colleagues working in the ICTD space have had mixed experiences with 3rd party SMS gateway providers in developing countries. So rather than follow the same approach, our end-to-end system leverages SMSSync [70], an Android application that adds SMS gateway functionality to the Android on which it is installed. In the new architecture, our web application is hosted as a cloud service that can communicate with monitoring devices deployed in the field via one or more in-country Android gateways. While we have not yet field-tested this approach, colleagues have confirmed that this approach worked well for them in developing countries.

Recharging the prepaid SIM cards being used in the trial was also a challenge because it required a person to travel to each facility and add airtime to the phones' SIM card. For larger scale operations, a centralized, postpaid billing system would work much better for cold chain staff.

The deployment in Albania also gave a new perspective to cold chain supervisors and researchers. The FoneAstra-based continuous monitoring system began to highlight the distinction between data meant for different stakeholders in the system. For instance, supervisors were most concerned about the long-term temperature history of equipment, while store managers and health workers were most concerned about the short-term operational conditions (e.g., current temperature, daily maximum/minimum, alarm status, etc.). Our system generates a lot of data; channeling the right data to the right stakeholders in a timely manner will be important moving forward.

## 4.5 Android-based Cold Chain Management

Android devices were expensive when we started our cold chain monitoring work back in 2010. The cheaper Androids cost over $150 USD at that time, making it impractical to use them in large-scale cold chain deployments. However, Android prices have dropped significantly in the past years, and today the cheaper Androids can be bought for less than $50 USD, making them more attractive to use in our work. Lessons learned from the Abania trial also indicated the need for a new monitoring device that has a better user interface and is more configurable than our previous system. Motivated by these factors, we are now develping a cold chain monitoring system for Android devices. This effort has also guided the development of ODK Sensors (cf.

Chapter 3) and FoneAstra 3.0 (cf. Section 2.3), which are both being leveraged in the new system.

The latest FoneAstra board can host up to four 1-wire temperature sensors and can be easily extended to host more such sensors via a simple adapter. Additionally, an Android device can communicate with multiple FoneAstras located within Bluetooth range. These capabilities make the monitoring system scalable enough to address the varying needs at different levels of the cold chain. On the Android side, ODK Sensors facilitates communication with FoneAstra(s). Sensor data received from FoneAstra is processed in a sensor driver implemented as a separate Android application (cf. Section 3.3.2.2). Much of the logic specific to cold chain temperature monitoring is implemented in a separate Android application that leverages ODK Sensors. For instance, this application lets users discover sensors connected to a co-located FoneAstra and assign equipment names to each sensor. It communicates with the backend server to upload configurations, alarm messages and routine temperature reports. It can also receive configuration commands from the server to adjust parameters like sampling interval and reporting rates. In the near future, ODK Tables [71], a tool in the ODK tool suite, will have access to the temperature and equipment data stored on the Android, enabling users to graph and visualize the data locally on the device.

Our new monitoring system can be deployed for cold chain monitoring using two alternate configurations:

1) In parts of the cold chain that have cellular coverage, Android phones coupled with FoneAstra(s) will be deployed to provide real-time temperature monitoring. This system will send real-time alerts and routine temperature reports to a central server.
2) In parts of the cold chain where cellular connectivity is either not available (e.g., remote faciltites or air/ocean transit routes), or if real-time monitoring is not needed (e.g., to lower operational costs), FoneAstras will be deployed in autonomous mode, i.e., without a co-located Android, to monitor and record temperatures locally. Cold chain staff will periodically connect to these FoneAstras (e.g., during routine visits to facilities) over Bluetooth via their Android phones to retrieve and visualize the stored temperature data and to upload the data to the server.

This system will provide a complete view of the cold chain to supervisors and could eventually lead to better cold chain management. Additionally, it can evolve to cover other aspects of cold chain management (e.g., cold chain logisitics) or integrate with other workflows at facilities (e.g., healthcare delivery at clinics). In the new system we have offloaded sensing to the battery-powered, Bluetooth-enabled FoneAstra. This makes it possible for the Android to be mobile, and potentially be used to address other needs of the facility, while real-time temperature monitoring and reporting happens in the background (as long as the Android is within Bluetooth range of FoneAstra(s)). Inadequate financial resources prevent Ministries of Health in developing countries from scaling up cold chain management technologies to cover their country-wide operations. However, the technology cost per facility will drop if we provide an integrated solution to address multiple aspects of cold chain management rather than deploying separate, single-purpose technologies. This could help lower the barriers to deploying country-wide cold chain management systems.

### 4.5.1 Evaluation

We have been testing our new system in the lab while it is being developed and are also looking for field-trial opportunities. In collaboration with a California-based non-profit organization, we got an opportunity to field-trial our system's second deployment configuration noted above. This partner routinely sends temperature-sensitive drugs and vaccines to Haiti via air or ocean freight. They rely on the pharmaceutical company's packaging to ensure that the shipments are always within the recommended temperature range (+2°C to +8°C) while in transit, but they wanted to verify this.

We included a FoneAstra with one temperature sensor connected to it in one of their FedEx Air shipments sent from California to Port Au Prince, Haiti. Medical supplies in these shipments are packaged in Styrofoam boxes and have cooling gel-packs to maintain an acceptable, low temperature inside the box. Since we had to place FoneAstra inside the box, where the moisture content would be high, we packaged it in a waterproof Pelican case, which was placed inside the Styrofoam box. This ensemble is shown in Figure 38. FoneAstra was programmed to sample its temperature sensor every 5 minutes and store the reading in its on-board memory card.


**Figure 38: A FoneAstra 3.0 board with its battery (which is not visible because it is under the board) placed inside a waterproof Pelican case. The three wires protruding from the left side of the board are for the temperature sensor that is plugged into the board. This ensemble was placed inside a Styrofoam box that contained temperature-sensitive medical supplies (along with cooling gel packs to maintain a low temperature) and was used to monitor the box's internal temperature while it was in transit, being shipped from California to Port Au Prince, Haiti.**

Figure 39 shows the temperature graph of the shipment based on data recorded by FoneAstra during the 14/02/2013 to 22/02/2013 time period. The package was shipped out of California on the 14th; FoneAstra was turned on and left at room temperature for some time before being placed in the Styrofoam box. The graph shows that the temperature started to fall shortly after FoneAstra was placed inside the box. However, something went wrong on the 15th, and the temperature started rising. The box reached Port Au Prince on the 15th and was awaiting customs clearance before it got delivered on the 18th. After delivery, the Styrofoam box, along with FoneAstra, was placed in a cold room, which explains the drop in temperature. FoneAstra was removed from the cold room after a short while and then turned off on the 22nd. This explains the rapid rise in temperature on the 19th and the temperature variations that occurred until FoneAstra was turned off. We were unable to determine the cause of the problem on the

15th or to find an explanation for the abnormal temperature curve between the 15th and the 18th, i.e., when the shipment was awaiting customs clearance.

This experiment identified a weak link in the cold chain that concerned our partners. It also highlighted the need for reliable and continuous temperature monitoring throughout the cold chain for temperature-sensitive medical products. While it is quite circumstantial, if temperature-sensitive drugs and vaccines are at risk of exposure to deviant temperatures while being transported to the country of delivery, and if such exposures go undetected, then monitoring the cold chain within the country would add limited value. Similarly, monitoring only some parts of a country's cold chain would add limited value as well.

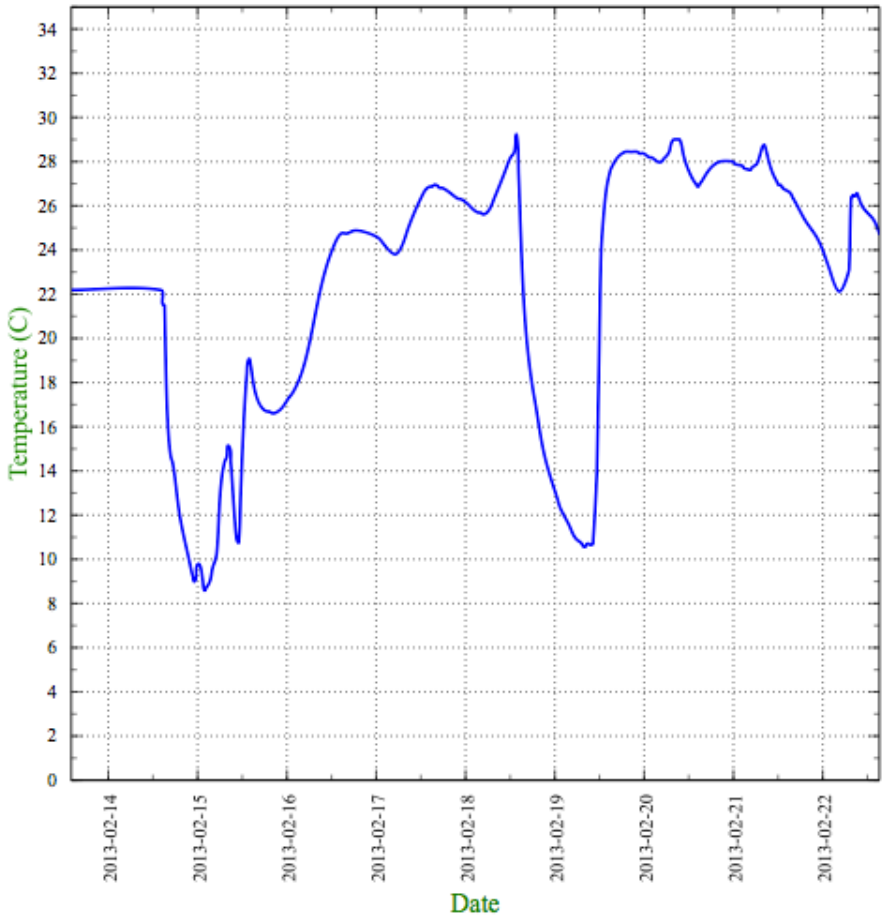



**Figure 39: Temperature curve of the Styrofoam box while it was en route to Port Au Prince, Haiti. FoneAstra recorded temperatures during the Feb 14th to Feb 22nd time period. The trip from California started on the 14th, and the temperature was dropping as expected. However, it started rising on the 15th due to a problem we could not identify. The package reached Port Au Prince on the 15th; it was at customs until delivery on the 18th, when it was placed in a cold room, which explains the subsequent drop in temperature. FoneAstra was taken out of the cold room on the 19th (note the rise in temperature around that time) and eventually turned off on the 22nd.**

## 4.6 Related Work

### 4.6.1 Temperature Loggers

Devices like FridgeTag [69] and LogTag [72] are simple temperature loggers that are placed inside fridges or cold rooms used to store vaccines. They can record temperatures for an extended period of time and provide some visual feedback. The low cost of these devices makes them appropriate for large-scale deployment in the cold chain. However, they do not provide real-time, continuous monitoring. Additionally, it is either difficult or not possible to retrieve historical data from such devices. FoneAstra configured to autonomously monitor and record temperature locally (Section 4.5) simplifies the retrieval of stored historical data over a Bluetooth connection, making it possible for such data to be retrieved using any Bluetooth-enabled mobile device.

### 4.6.2 Real-time Temperature Monitoring Systems

More expensive real-time, continuous monitoring systems like Cold Cloud (formerly Beyond Wireless) [73] and Vaccine Guard [74] are also used for cold chain monitoring. These are based on cellular modems that communicate with a backend to provide real-time monitoring. Due to their high cost (one-time device cost and recurring service cost), they are more appropriate for deployment only at higher levels of the chain that store large volumes of vaccines.

ColdTrace [75] is a low-cost, Android-based temperature-monitoring device that comes closest to our Android-based cold chain monitoring system. However, ColdTrace is a single-sensor solution (similar to Vaccine Guard that has one sensor per device) that connects the temperature sensor to the mobile device via the audio headset port. Additionally, since sensor sampling and analog-to-digital conversion of sensor data is done on the Android, it is not as energy efficient as our approach, where these features are implemented on a low power microcontroller.

## 4.7 Conclusion

Vaccines are biological substances that must be stored in a temperature-controlled environment, typically +2°C to +8°C. The temperature-controlled supply chain used to store and transport vaccines in a country is known as the “vaccine cold chain”. The cold chain infrastructure in developing countries is particularly weak; problems such as unreliable

infrastructure and faulty equipment often expose vaccines to temperatures that are either too high or too low, which can lead to vaccine spoilage. This chapter reported about our work on mobile phone-based, real-time, continuous temperature monitoring systems for vaccine cold chains.

Our first cold chain monitoring system was based on FoneAstra and low-tier mobile phones. In collaboration with PATH, WHO, and the Albanian Ministry of Health, we did a 6-month trial of this system in Albania in 2010 that monitored various parts of the country's cold chain.

Experience gained from the deployment in Albania helped us better understand the monitoring requirements at different levels of a cold chain. As a result, we are now building an Android-based monitoring system that is more interactive and will scale better to meet these varying requirements. We also re-architected the end-to-end system, enabling the server to be hosted anywhere in the cloud and communicating with in-country monitoring devices via an Android-based SMS Gateway that is also deployed in-country.

# Chapter 5

# Human Milk Banking

This chapter presents a mobile sensing system we developed to manage the pasteurization of donor breast milk at human milk banks. This work is being done in collaboration with PATH [20], a Seattle-based non-profit organization, and the human milk banking Association of South Africa (HMBASA). This chapter provides additional evidence for contribution 3 of the thesis: **Implementation and evaluation of sensing systems to enhance workflows in low-resource settings.**

We start by describing the need for human milk banking, which can help lower the newborn mortality rates in developing countries. The human milk banking process is described in Section 5.2, while Section 5.3 discusses our pasteurization management system and its evolution over the course of the project. In-lab experiments done to validate the system prior to field trials are presented in Section 5.4. Section 5.5 describes our deployment experience when installing the system at several milk banks in South Africa. We discuss related work in Section 5.6, and conclude the chapter in Section 5.7 with a discussion about future directions for this work.

## 5.1 Motivation

It is estimated that 3.3 million newborn babies die within the first month of life [76], and overall, 7.3 million children under the age of five die each year [77] in developing countries. According to a report [78] from the Lancet Child Survival Series, up to 13% of the deaths of children under five could be prevented by breastfeeding alone. Breastfeeding is considered to be a pillar of child survival due to its well-documented immunological and nutritional properties. Infants in developing countries who are fed formula in lieu of breast milk are at increased risk of sickness and death since they are often exposed to potential pathogens through unsafe water used in the formula and unhygienic conditions [79]. The World Health Organization recommends

providing donor breast milk when mothers' own breast milk is not available [80]. However, providing safe breast milk to infants in developing regions continues to be a challenge.

The availability of donor breast milk to vulnerable infants (those born pre-term, with low birth-weight, to HIV-positive mothers, or orphaned at birth) can be increased significantly by establishing human milk banks. Brazil's accomplishment with human milk banking is noteworthy, where the government has integrated milk banking within the national health system. Over the years 1980 – 2009, Brazil has lowered its infant mortality by 50%, and donor milk saves the public sector an estimated $540 million annually in reduced medical costs [80].

**Figure 40: A high-end commercial pasteurizer. It provides temperature monitoring and archiving on a computer via a temperature probe that is connected to the computer. Milk bottles placed in the pasteurizer are shown in the inset. Photo credit: iThemba Letu transition home, Durban, South Africa.**

Pasteurizing donor milk is a critical component of human milk banking. Milk from donor mothers must be pasteurized to ensure that pathogens are destroyed, but nutrients and immunological benefits preserved, before it is fed to infants. Human milk banks in developed countries often use commercial pasteurizers, like the one shown in Figure 40, to accomplish this. While such pasteurizers ensure that milk is never under-heated and minimize over-heating, they are expensive (up to $50,000 USD) and require reliable infrastructure (e.g., electricity, clean water, connection to a computer for recording pasteurization temperatures), which makes adoption difficult in low-resource milk banks in developing countries. Additionally, since these

pasteurizers process large volumes of milk at a time, banks often freeze donor milk until a large enough batch of milk-bottles can be processed. Likewise, after a large batch of milk is pasteurized, not all of it may get fed to babies immediately, and the unused milk is frozen again. Repeated freezing of milk, especially before pasteurization, degrades its nutrients.

Flash heat pasteurization [81] has been proposed as a low-cost, low-tech alternative for pasteurizing breast milk in low-resource settings. In this method (set up shown in Figure 41), a glass jar containing breast milk is placed in a pan of water (the water-bath), which is then heated. When the water-bath reaches a rolling boil, milk achieves a high enough temperature (about 72°C), to ensure that contaminants are destroyed. At this point, the milk is removed from the water-bath and cooled before being fed to a baby. This method was originally designed for in-home use by HIV-positive mothers to pasteurize their own breast milk to feed their babies. Due to its practicality and affordability, this method has been implemented at some milk banks in South Africa. However, it lacks adequate temperature monitoring, and the interpretation of "rolling boil" of the water-bath is subjective. For large-scale implementation, a more rigorous monitoring system is needed, one that provides the appropriate process-control and quality assurance.

**Figure 41: Low cost, low-tech flash heat pasteurization being used at a human milk bank. A milk jar, without a lid, is placed in a pan of water, which is then heated. When water reaches a rolling boil, milk reaches a temperature high enough to deactivate contaminants like HIV and other pathogens. Photo credit: King Edward Hospital, Durban, South Africa.**

Figure 42 (reproduced from [81]) shows the temperature curve of 50 mL of milk in a glass jar while it was being heated in a pan containing 450 mL of water. The peak temperature of

about 72°C was reached within five minutes, and the jar was removed from the water-bath immediately thereafter to cool the milk.

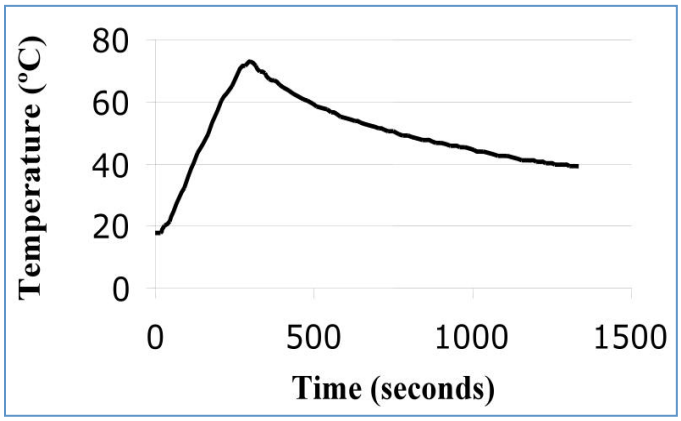



**Figure 42: Typical temperature curve of milk during flash heat pasteurization. Milk is heated rapidly in a water-bath, then cooled down.**

The Department of Health (DOH) in South Africa wants to significantly scale-up human milk banking across the country. To do so, they are looking for affordable methods to safely pasteurize human breast milk. They need a system that can provide adequate temperature monitoring to enable better control of the pasteurization process. Additionally, they want the system to generate proper records for the procedures performed; this provides the much needed quality assurance and lets supervisors remotely monitor operations at facilities under their jurisdiction.

To address the needs of the DOH, we developed a mobile sensing system to manage the pasteurization process at human milk banks. The total cost of our system is under $600 USD, and it includes a mobile phone, FoneAstra, a Bluetooth-enabled printer, an induction stove, a jar-holder and glass jars. The work presented in this chapter is a multi-disciplinary collaboration among teams with backgrounds in engineering, maternal and child health, and commercialization. It reflects our evolutionary approach to developing appropriate technologies to address problems in developing regions. Lessons learned after building an initial prototype guided the development of the solution that we trialed in the field. Additional feedback led to further system refinements, making the system more appropriate for end-users' needs. As 2014, our system has been installed at five locations in South Africa, with plans to scale up further.

## 5.2 The Human Milk Banking Process

Interviewing milk-banking stakeholders in South Africa gave us a clear picture of the human milk banking process, depicted in Figure 43. Staff screen and recruit lactating donor mothers from neonate wards in hospitals or their communities. Screening involves checking for existing conditions like HIV, hepatitis, prescription drug use, etc. Once recruited, mothers are encouraged to donate excess breast milk to the milk bank. The first time a mother donates her milk, a sample from the milk (before pasteurization) is sent to a microbiology lab to test for microbial activity (this optional step is not shown in Figure 43). If test results indicate that microbial activity is below a certain threshold, and certain heat-resistant pathogens are not found (e.g. bacillus), milk is considered safe, and is pasteurized. Samples are taken from post-pasteurized milk and sent to a microbiology lab to confirm that any pre-existing pathogens have been deactivated and that there has been no post-pasteurization contamination. Samples that pass post-pasteurization microbiology tests are safe to be fed to babies and are either frozen and stored for future use or used immediately.

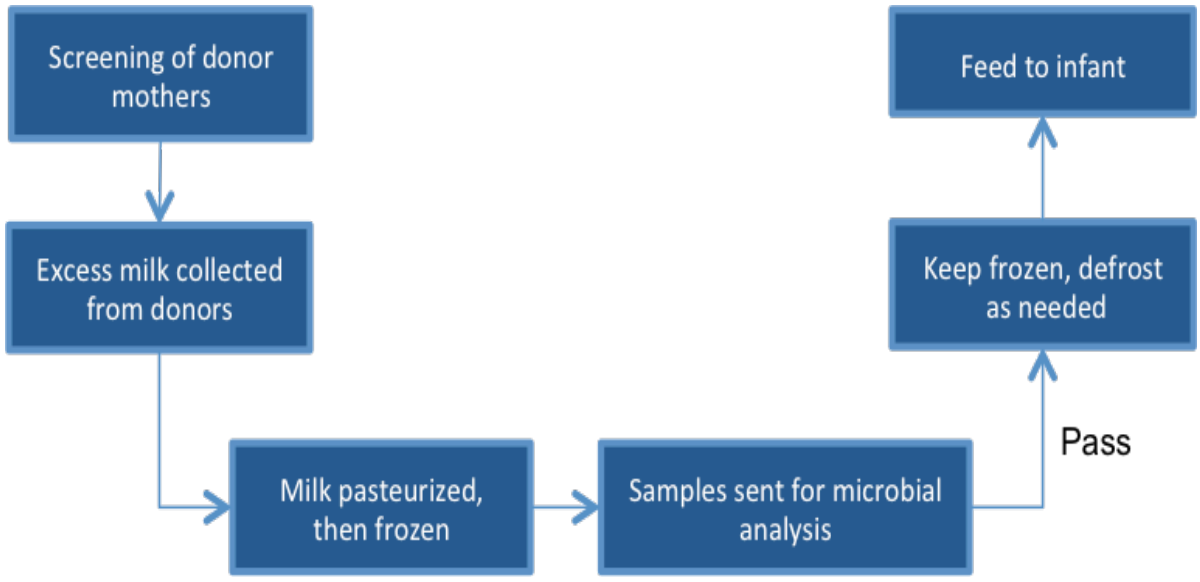



**Figure 43: The human milk-banking process. Image credit: PATH, Seattle, USA.**

Countries like Brazil follow a centralized model for human milk banking. In this model, milk banks are established across the country (Brazil has 200 such banks). Each bank can have several pasteurizers that process large volumes of milk. Donor milk from several organizations (e.g., postnatal care facilities at hospitals, clinics, etc.) is sent to the milk bank for pasteurization. After pasteurization, the milk is sent back to the organization from which it came and is fed to

babies. This model is relatively easy to administer from the perspective of a state or a central government; however, the capital investment required is high. Additionally, vulnerable infants in remote, rural areas are often beyond the reach of centralized milk banking systems.

A decentralized model for human milk banking is less common in countries today. In this model, hospitals run their own milk banks to cater to the infants in their neonate wards. This model is relatively easy to set up and operate. The capital investment is lower and more appropriate for low-resource settings; however, it is difficult to manage and monitor by supervisors due to its decentralized nature.

We believe that provision for breast milk for vulnerable infants can be maximized in a country if both of these models co-exist. The work presented in this chapter provides quality control and quality assurance for milk banks at an affordable price point; this will help strengthen, and eventually scale-up, decentralized human milk banking in countries. A better understanding of the milk banking process helped us realize the need for a unified information management system for milk banking that manages information needs for the entire process. Our work thus far is a first step toward building tools and systems that simplify a government's ability to administer decentralized human milk banks.

## 5.3 Mobile Device-based Milk Pasteurization Monitor

To provide better quality control, we added mobile phone-based, real-time temperature monitoring to the flash heat pasteurization process. This enables technicians at human milk banks to better control the procedure. Our system monitors temperatures continuously and provides appropriate feedback to the technician at different stages of the procedure; this helps technicians better control the process. At the end of the procedure, the mobile device lets the technician print labels for milk-bottles via a Bluetooth-enabled printer, and it also uploads the temperature data to an Internet-accessible server for archiving. Supervisors access the server from an Internet browser to view the archived data and verify that milk pasteurization is being done correctly at their milk banks. This section discusses the iterations of our monitoring technology, and how it has evolved over time to integrate better with the human milk banking workflow.

### 5.3.1 Usability Assessment and Pasteurization Monitor V1

Since we had already developed a temperature monitoring system for vaccine cold chains (cf. Chapter 4), we initially chose to leverage the same platform (i.e., FoneAstra with a temperature sensor coupled to a low-tier mobile phone) for monitoring flash heat pasteurization. However, unlike cold chain monitoring, where the system operates autonomously, the milk pasteurization application requires a human to be in the sensing loop and to act upon feedback received from the monitoring device in real-time. This made it important for us to understand the contextual settings in which the device would be used and to evaluate the output modalities available on our monitoring device to determine the most effective feedback mechanisms.


**Figure 44: Pasteurization monitor V1: The mobile phone and FoneAstra are inside the blue enclosure. FoneAstra was enhanced with an LCD, water-proof temperature sensor, and an audio buzzer.**

Feedback to the user was important in this application; therefore, we enhanced the output capabilities of FoneAstra to make it more suitable for this application. Particularly, we switched

to using larger, more prominent LEDs and added an audio buzzer and a 16-character, 2-line LCD to FoneAstra. We found an off-the-shelf, food-grade, waterproof 1-wire temperature sensor (the same type of sensor used in the vaccine cold chain monitoring project, so the sensor interfacing code did not change). This first version of the pasteurization monitor is shown in Figure 44.

| | Light | Sound | Text |
|---|---|---|---|
| **Heating milk** |  | No beep | "Heating. " + current temperature |
| **Heating complete** |  |  | "Cooling. " + current temperature |
| **Procedure complete** |  | No beep | "Procedure complete." |

**Figure 45: An example of a table shown to survey participants regarding the most effective way to provide feedback from FoneAstra. A dotted line under the "Light" column indicates a blinking LED, while a solid line indicates that the LED is turned on. Color of the line indicates the color of the LED. A dotted line under the "Sound" column indicates a beeping sound.**

We learned from staff at milk banks in South Africa that flash heat pasteurization is done in special rooms that would have ample space for mounting the pasteurization monitor. The staff are either trained nurses or volunteers who receive basic training to perform the procedure. It was important to build a system that was simple to use and gave clear feedback to effectively guide users through the process. We surveyed colleagues at the University of Washington to determine the most effective means to provide feedback to the user performing pasteurization. Survey participants were shown 15 different tables (similar to that shown in Figure 45), and each table listed the state of each output modality (table columns) at different stages of the pasteurization process (table rows). We had 26 participants in this survey, and Figure 45 was the most popular choice of outputs amongst the participants. We also validated this choice with milk bank staff in South Africa before implementing it on the device.

The software elements of FoneAstra were based on FreeRTOS [19] (like our cold chain monitoring system). In this version, FoneAstra's temperature probe was meant to be placed in a

glass jar containing breast milk, which was placed in a water-bath. The user turned FoneAstra on and started heating the water-bath. The temperature sensor was sampled every second, and the temperature was displayed on the LCD. The green LED blinked until the pre-configured high temperature threshold of 72°C was reached. At this point, FoneAstra would start beeping and the red LED would start blinking, while the green LED was turned off. This told the user to stop heating and start cooling the milk. Temperature monitoring continued while the milk cooled down, and the procedure was considered complete when the temperature dropped to 25°C. At this point, the beeping would stop, the red LED would turn off, and the green LED would turn on. FoneAstra sent the entire pasteurization temperature curve to the server via several SMS messages.

Although the user interface required more work (specifically, we were not displaying text per Figure 45 on the LCD), we started testing this device in Seattle. A PATH collaborator also took the device to South Africa to get feedback from HMBASA, our in-country partner. They approved of the device and liked its simplicity and feedback mechanisms. However, our PATH colleague brought back new information about the human milk banking process; this led us to consider a more capable mobile device that would allow additional data to be recorded digitally (e.g., information about breast milk donors). We therefore started exploring options for creating a smartphone-based pasteurization monitor, described in the sections below.

### 5.3.2 Pasteurization Monitor V2

The next version of the pasteurization monitor was developed as an Android application that leveraged ODK Sensors (Chapter 3) and a USB Bridge (Section 2.2.2). We integrated the 1-wire temperature sensor with the USB Bridge, and software running on the Bridge sampled the sensor every second and sent the reading to the Android over the USB connection. An ODK Sensors driver parsed the sensor data and provided this data to the pasteurization monitoring application.

Figure 46 shows screenshots of the pasteurization monitoring application that provide guidance to the user. When the user clicks on the “Start” button, the application connects to ODK Sensors, which in turn establishes a connection session with the USB Bridge. After this, the application queries ODK Sensors for temperature readings every second and provides audiovisual feedback based on the readings to guide the user through the pasteurization process.

At the end of the procedure, the application uploads the temperature data to a server that is hosted on the Amazon Cloud. This data is accessible to supervisors over an Internet browser.


**Figure 46: Screenshots of the Android application that guides the user through the pasteurization process. (1) User places water and milk jars in the water-bath and turns the stove on. Clicking the "Start" button on the app starts the pasteurization monitoring process. (2) A unique batch ID is generated for the procedure. Rising temperature is displayed on the screen along with the time elapsed. (3) Phone beeps twice to alert user at 5°C before the high temperature threshold of 73°C is reached, and the visual indicator changes accordingly. (4) The visual indicator changes at 73°C and the phone starts beeping continuously until the user removes the jars from the water-bath. (5) Beeping stops, and the visual indicator changes to indicate cooling when app detects a drop in temperature. (6) Procedure is complete when temperature drops to 25°C.**

Figure 47 shows a screenshot of an archived temperature curve accessible from the server. In the previous version of the pasteurization monitor, SMS was the only communication mechanism available to upload data to the server, and each pasteurization procedure resulted in over ten SMS messages to upload the temperature data. This worked out well in our lab tests;

however, the reliability of SMS is highly dependent on a service provider's network. In our new system, we switched to GPRS/3G for uploading data to the server since it is more reliable than SMS.

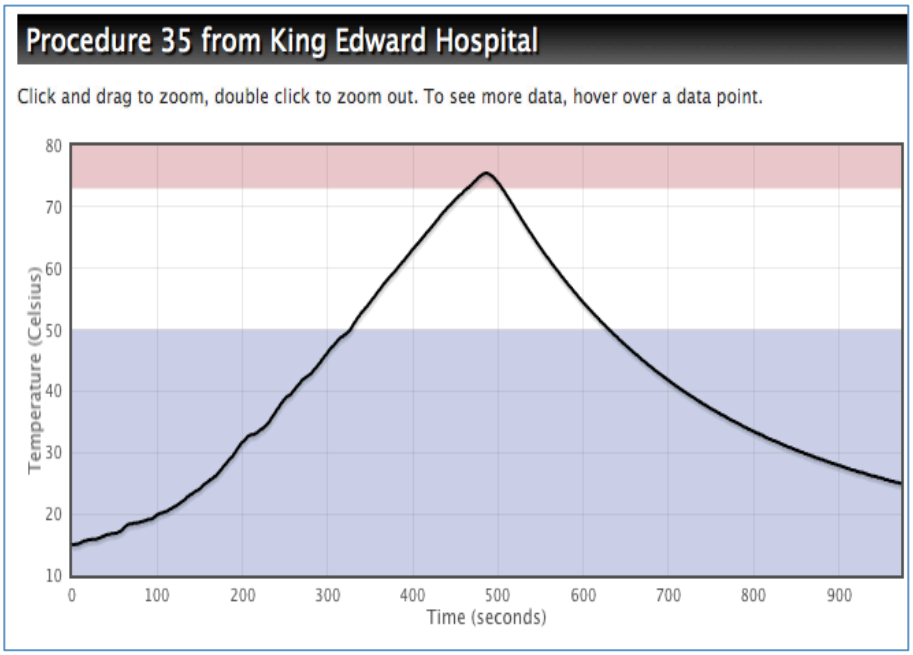



**Figure 47: Archived pasteurization temperature curve from the server. This is accessible to milk bank supervisors from a web browser.**

A smartphone-based platform let us quickly address new requirements we identified in the field. Section 5.5.2 describes the enhancements we implemented during the deployment period itself. We also changed the new system to require that the temperature probe be placed in a jar containing water, not breast milk; the temperature of the water jar thus controlled the pasteurization process. We did this to eliminate any chance that the temperature probe would contaminate the milk that got pasteurized. Additionally, it afforded the practical benefit of eliminating the need to sterilize the temperature probe before each procedure. In this approach we leveraged the fact that milk and water have very similar specific heat capacities; i.e., given equal volumes and the same amount of heat – their temperature rise will be similar. Experiments to validate this approach are described in Section 5.4.3.

To ensure that the water and milk jars got the same amount of heat, engineers at PATH built a stand to hold the jars (shown in Figure 48). The stand was placed in a water-bath that was heated on a stove. Additionally, we required that the volumes of milk and water in all jars be the

same. In our setup, this volume was 120mL. With this volume and the jars used in our system, the liquid reached the upper lip of the jars. While jars of some other capacity could also be used in our system, the jars we chose let users easily ensure that each jar had the same volume of liquid by visual inspection, without the need for measuring apparatus. We also required that the initial temperature of water and milk be approximately the same in order to correctly determine when milk reached the high temperature threshold. If milk were thawed and cold, users were required to use refrigerated water; otherwise, for freshly expressed breast milk, tap water at room temperature was required.


**Figure 48: Pasteurization monitor V2. A stand holds 4 glass jars. The temperature probe connected to the mobile phone is secured on the jar containing water; the other jars contain milk. The stand with jars containing liquid is placed in a water-bath that gets heated on an induction stove. The white enclosure on the left side of the picture hosts the mobile phone and the USB Bridge that connects the probe to the phone. Photo credit: Pediatrics Lab, University of KwaZulu-Natal, Durban, South Africa.**

Figure 48 shows the milk pasteurization system that we took to Durban in the summer of 2012. It had one jar with water that has the temperature probe for monitoring pasteurization temperatures and three jars with milk (our system could pasteurize one to three jars of milk at a time). The stand with jars was placed in a water-bath that was heated to pasteurize the milk. We used an induction stove for heating since it was a more efficient source of heat compared to an electrical stove, although any source of heat is acceptable. The mobile phone that ran the pasteurization application and the USB Bridge connected to it were packaged together in an enclosure. The temperature probe connected to the USB Bridge was attached to the water jar's lid.

While we made several enhancements in this system version (see Section 5.5.2 for other enhancements), some issues remained that posed barriers to scaling up the system for widespread deployment. Specifically, the Android mobile phone communicated with the temperature probe via a USB bridge that was not battery-powered. This was acceptable for deployments at neonate wards in hospitals where electricity was generally stable; however, it was not suitable for environments with unreliable electricity. Additionally, communicating with a USB bridge required that the mobile phone implement the Android Open Accessory protocol [44], typically available only on newer, higher-end Android devices. This raised the cost of the solution. We addressed these issues in the next iteration of the system.

### 5.3.3 Pasteurization Monitor V3

Limitations of the previous version led to the design of FoneAstra 3.0 (cf. Section 2.3), a battery-powered, Bluetooth-enabled sensor interfacing board that acts as a bridge between the temperature probe and the mobile phone. Switching to Bluetooth let us use any Bluetooth-enabled mobile phone (although we continue to use Android phones), giving us more flexibility at a lower price-point. Swapping out the USB Bridge with the new FoneAstra was fairly straightforward. We just implemented code for FoneAstra that sampled the sensor every second if there was a Bluetooth connection and sent the sensor data over that connection. On the Android side, the pasteurization monitoring application did not change at all. We reused the same sensor driver and only had to modify its metadata to tell ODK Sensors that this driver gets sensor data over Bluetooth instead of USB. We visited Durban again in November 2012 to upgrade the existing systems and establish a few more milk banks there and in Cape Town. Figure 49 shows the new system in use at a milk bank in Cape Town.


**Figure 49: Pasteurization monitor V3 installed at a human milk bank in South Africa. It uses the FoneAstra 3.0 board (cf. Section 2.3) to interface the temperature sensor to Android over Bluetooth. FoneAstra is inside a black enclosure mounted on the wall, and the Android phone is mounted on the enclosure. The inset shows a close up view of the ensemble. The rest of the set up (e.g., stand, pan, stove etc.) is the same as in V2. Photo credit: Paarl Hospital, Cape Town, South Africa.**

## 5.4 In-Lab Experiments

This section discusses some in-lab experiments we performed before deploying our system in the field. These experiments helped us understand the impact of the source of heat on the pasteurization process and validate certain aspects of the system.

### 5.4.1 Impact of Heat Source

Flash heat pasteurization aims to heat milk to the high temperature threshold (approximately 72$^{o}$C) as fast as possible and then cool it down before storing or using it. In traditional flash heat pasteurization (i.e., without temperature monitoring), the water-bath

reaching its boiling point (100$^{o}$C), indicated by a “rolling boil”, is used as the visual cue to indicate that the high temperature threshold has been reached and the milk should be cooled down. While earlier work has shown this indicator to be accurate in controlled environments ([81]), we expected that as human milk banking is scaled up, the type of heat source used and environmental conditions (and other factors) could vary. Therefore, we wanted to understand how varying intensities of heat affected the pasteurization process. This section presents the results from our in-lab experiments where we simulated flash heat pasteurization at varying intensities of heat.

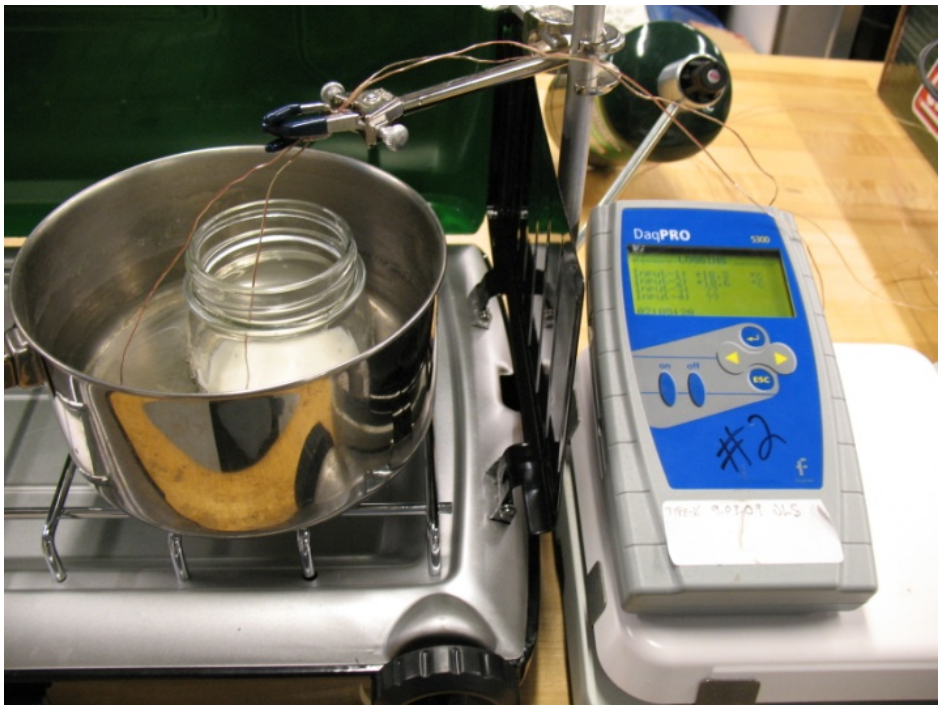



**Figure 50: Experimental setup to understand the impact of the heat source. A glass jar containing milk was placed in a water-bath and heated on a stove. Two thermocouple probes connected to a DaqPRO data logger monitored the temperature of milk and water, respectively. Photo credit: PATH, Seattle, USA.**

Our experimental setup, shown in Figure 50, resembled the setup used in previous studies ([81], [82]) with an off-the-shelf propane gas stove as the source of heat. A pan with 450 mL of water was used as the water-bath, and 50 mL of whole milk in a glass jar was used in lieu of breast milk (whole milk has thermal properties close to that of breast milk). Two thermocouple temperature sensors connected to a DaqPRO data logger [83] monitored the temperature of water and milk, respectively. The data logger was configured to sample the sensor every second.

#### 5.4.1.1 *Moderate Heat*

Figure 51 shows the temperature curve of milk and water when a medium-intensity flame was used on the stove. This scenario replicated the results from previous studies. Milk reached 72°C at about the same time as the water reached its boiling point. The time period marked by the vertical, dotted lines in the figure represents the 15-second interval during which water was boiling and milk was in the 72°C temperature range. Milk was removed from the water-bath after this and allowed to cool down at room temperature (the sharp drop in water temperature is because the sensor was removed from water).

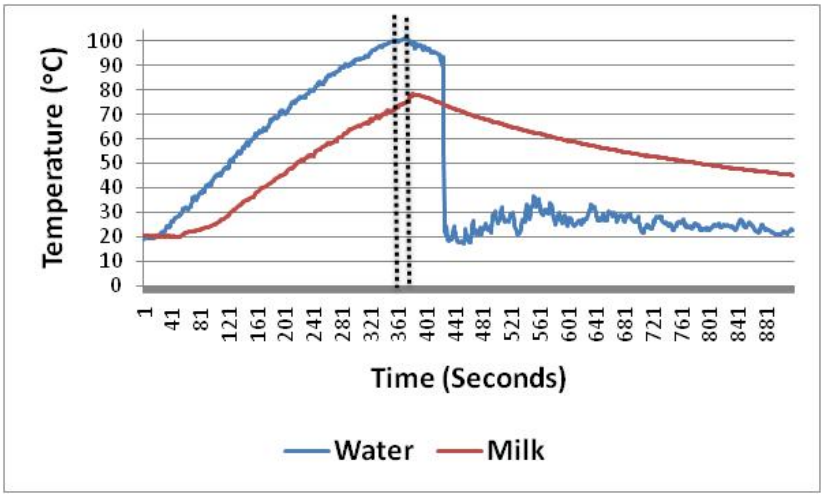



**Figure 51: Temperature curve obtained on a medium intensity flame. Milk reached 72°C at about the same time as the water-bath reached its boiling point of 100°C. Milk was removed from the water-bath 15 seconds after it reached 72°C. The 2 vertical dotted lines mark this 15-second period.**

#### *5.4.1.2 High Heat*

Figure 52 shows the temperature curve of milk and water when a high-intensity flame was used on the stove. Water started boiling at T=184 seconds; however, milk reached 72°C at T=250 seconds (indicated by the black arrow). The vertical, dotted lines mark the 15-second interval during which water was at its boiling point; milk was actually in the 55°C - 60°C range during this time and was removed from the water-bath 15 seconds after it reached 72°C. This experiment showed that if a high intensity heat source were used in real settings, and if water reaching its boiling point were used as the visual cue to remove milk from the water-bath, milk might not get completely pasteurized. If milk is not heated sufficiently, there is a risk that pathogens might not be destroyed during pasteurization.

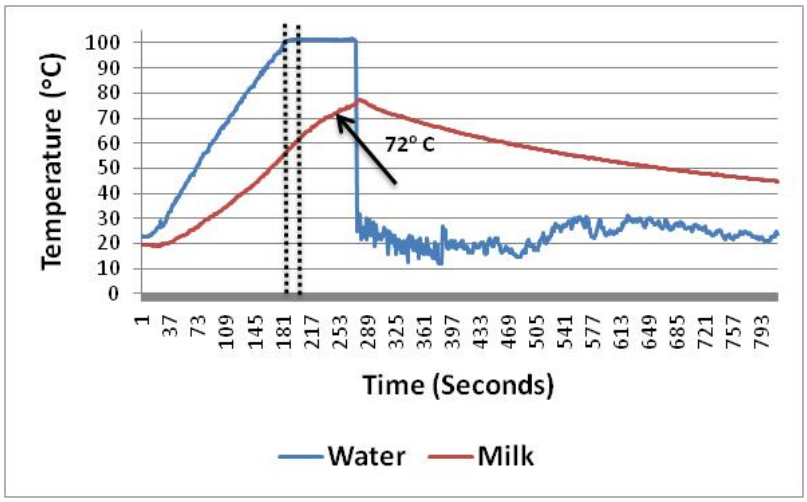



**Figure 52: Temperature curve obtained on a high intensity flame. The water-bath reached its boiling point very quickly (T=184), while milk reached 72°C later (T=250, indicated by the arrow).**

#### *5.4.1.3 Low Heat*

Figure 53 shows the temperature curve of milk and water when a low-intensity flame was used on the stove. In this case, water never reached its boiling point, while milk reached 72°C at T=1015 seconds (indicated by the arrow). Milk was eventually removed from the water-bath when the milk and water temperatures started to converge at around 80°C. In this case, then, there is a risk of overheating the milk and thereby lowering its immunological and nutritional properties.

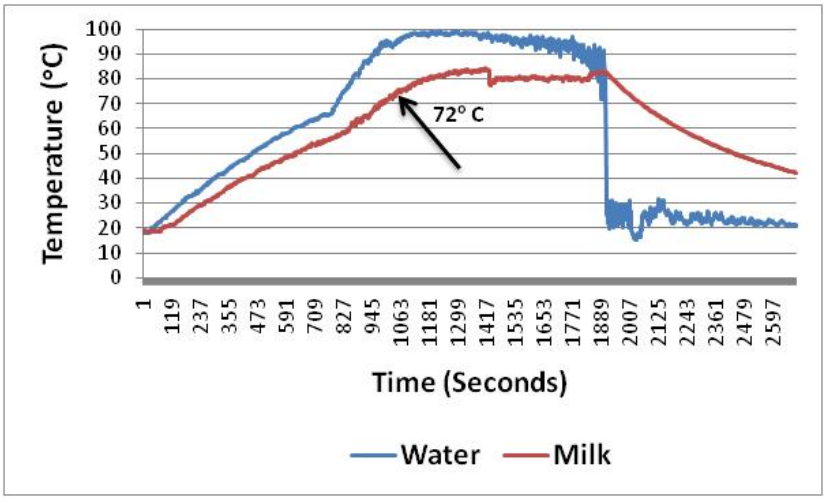



**Figure 53: Temperature curve obtained on a low intensity flame. The water-bath never reached its boiling point, while milk reached 72°C at T=1015 (indicated by the arrow). Milk was eventually removed from the water-bath and cooled down.**

We saw similar results when we performed experiments using electric and induction stoves as the heat source. Based on these experiments, we concluded that either continuous temperature monitoring should be a necessary part of flash heat pasteurization, or additional guidelines must be defined to ensure that water reaching its boiling point is a reliable indicator to control the pasteurization process. Moreover, the properties of the vessel used for the water-bath could shift these curves to the left or right, making it more difficult to predict the outcome from visual observations alone. These results strengthened the case for providing more comprehensive protocol instructions, such as specifying the type of heating source, vessel, etc., as well as monitoring temperatures for quality assurance in low-tech pasteurization systems.

### 5.4.2 Probe Validation

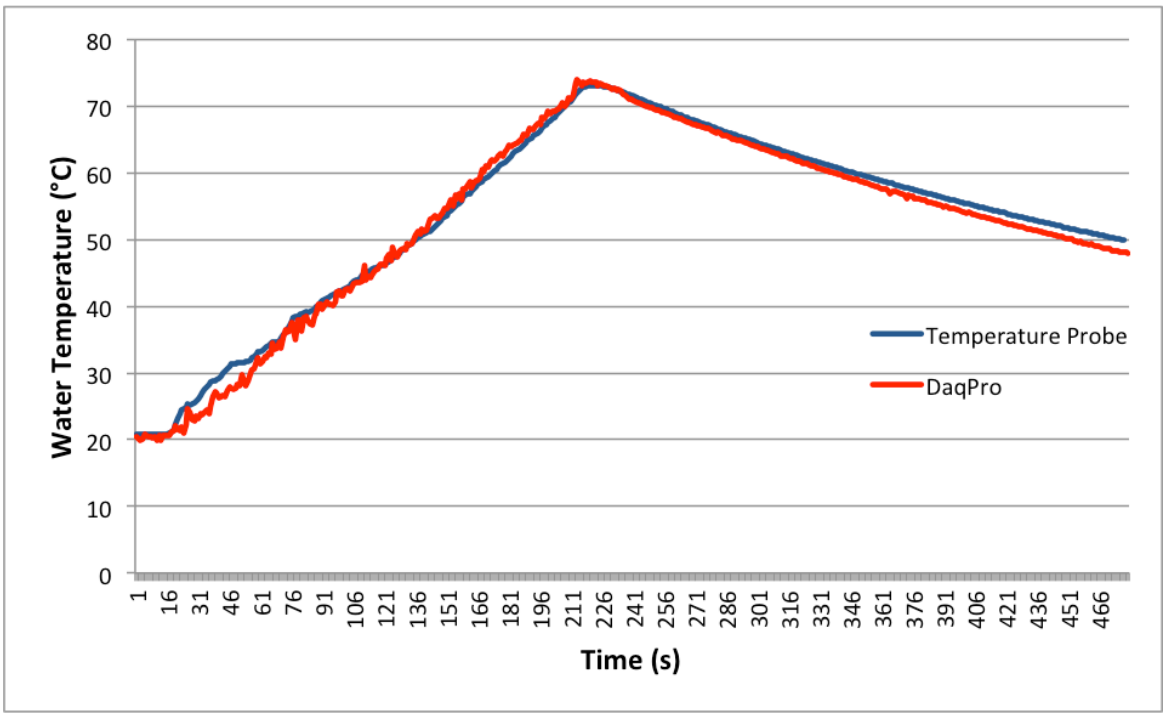



**Figure 54: A graph showing the responsiveness of the waterproof temperature probe (blue) compared to a DaqPro thermocouple (red). The average temperature difference between the two sensors is 0.68 °C in this experimental run.**

We used a waterproof temperature probe [84] that had a 1-wire digital temperature sensor encased in a one-inch long stainless steel tube. The short tube ensured that it got completely immersed in water to provide accurate readings. To ensure that the probe was responsive enough to temperature changes, we validated its readings against a DaqPro logger with a thermocouple

that had a fast response to temperature changes. In these experiments, the waterproof temperature probe and thermocouple were immersed in a jar containing water, which was then heated. The temperature variation between the two sensors averaged over 20 experimental runs was 0.71°C, which convinced us that our temperature probe was responsive enough for the application. Figure 54 shows the temperature curve of the two sensors from one of the experiments in which the average temperature difference was 0.68°C.

### 5.4.3 Validating the Use of Water to Control Pasteurization

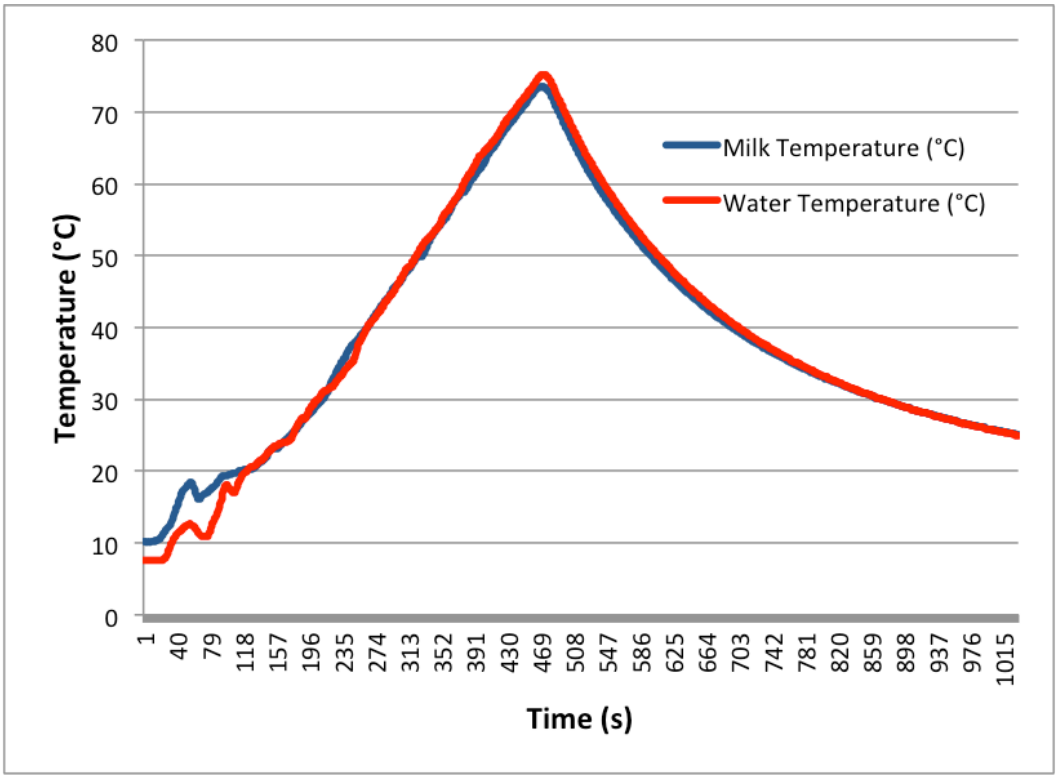



**Figure 55: A graph showing the temperature curve of water control (red) and milk (blue). When milk reached 72°C, water's temperature was in the 71°C to 73°C range.**

Our system sought to ensure that milk reached the target temperature of 72°C in each procedure. As mentioned in section 5.3.2, we monitored the temperature of a jar of water to control pasteurization. To validate that doing so still ensured that milk got heated sufficiently, we ran experiments to monitor the temperature of both the water and milk jars while they were being heated. Over 20 runs, we found that when milk's temperature reached 72°C, water's temperature was in the 71°C to 73°C range. Based on this, we set the high temperature threshold in our

system to 73°C. This ensured that milk always reached at least 72°C, a temperature high enough to deactivate pathogens.

Figure 55 shows the temperature curves of milk and water from one of the experiments. In this experiment, water achieved a peak temperature of 75.2°C, while milk achieved a peak temperature of 73.5°C. Our system erred towards slightly overheating milk (by 1-2°C for a few seconds) because it was safer than the prospect of under-heating, and potentially not completely deactivating pathogens. According to our project partners, experts in infant nutrition and milk banking, the benefits of using water to infer milk temperature to control pasteurization far outweigh the downside of slightly overheating milk.

### 5.4.4 User trial at PATH

Before taking the system to South Africa, we conducted a user trial in Seattle with eight subjects. All subjects were employees at PATH and were vaguely familiar with our project, although they had never pasteurized milk or seen our system before. They were given verbal training about the pasteurization process and our system and then asked to perform a procedure on their own. We intentionally did not provide intensive training because we wanted to understand the system's ease of use. Participants were encouraged to think aloud and verbally communicate any concerns they had across during the procedure. This proved to be a helpful exercise since we saw that all but two subjects erroneously ended the procedure on the mobile phone before the process was actually complete. This caused the mobile application to upload incomplete temperature curves to the server, while in reality the milk was cooling down. We addressed this problem immediately after the subject tests.

## 5.5 Deployment in South Africa

We travelled to Durban, South Africa for a 3-week period in May-June of 2012 to deploy our pasteurization monitoring system (V2 at that time). Our in-country partners – the Human Milk Banking Association of South Africa (HMBASA) and faculty at the Nelson Mandela School of Medicine at the University of KwaZulu-Natal (UKZN) – facilitated this deployment. The UKZN medical school is co-located with the King Edward Hospital (KEH), which has a human milk bank in its Neonate Intensive Care Unit (NICU). Under the guidance of HMBASA, this milk bank used traditional flash heat pasteurization to process donor breast milk for babies in the NICU.

Our goal was to introduce the new pasteurization system at two locations: (1) the milk bank at KEH for doing routine pasteurizations, and (2) the pediatrics lab at the UKZN where researchers would pasteurize donor milk and send samples for microbial activity analysis to validate the safety of our system. We set up our base at the pediatrics lab in UKZN, where we would demonstrate the system and train users. Since the milk bank at KEH already had a working system in place, we did not want to disrupt that until milk bank technicians were properly trained and comfortable using our system.

### 5.5.1 Baseline Data Collection

During the first week of the deployment, we visited a few milk banks (including the one at KEH) and conducted baseline interviews with key milk-banking stakeholders. With these interviews, we wanted to understand milk banking procedures and stakeholders' perception of these procedures and to identify opportunities where appropriate technology could be introduced to address gaps. The stakeholders included two milk bank supervisors (colleagues at UKZN), four milk bank technicians, and a neonatologist. One supervisor was a pediatrician, while the other was a lactation consultant who had been practicing for over ten years. Except for the two supervisors, none of the other stakeholders were aware of our new pasteurization system, and we kept this information from them until we completed all baseline interviews and started user training. The questionnaire used for these interviews is available in Appendix B.

All interviewees indicated that it was a challenge to maintain an active, large-enough pool of lactating donor mothers. The milk banks that use flash heat pasteurization typically pasteurize only when fresh donor milk is available. Depending on the availability of donor mothers, this translates to a few procedures per day. Pasteurized milk is then either fed to babies immediately or frozen for storage. One milk bank technician worked with a donated, commercial pasteurizer that processes large volumes of milk at a time. Thus, this technician collected and froze donor milk throughout the week and pasteurized a large batch of milk once a week. Even though she used a commercial grade pasteurizer that automated much of the procedure, it was more labor intensive for her because she needed to thaw the frozen, unpasteurized milk bottles before putting them in the pasteurizer. Most of the milk bottles were frozen again after pasteurization because the large volume of milk could not be consumed immediately. In addition to being labor intensive, we also learned from supervisors and doctors that repeated freezing and

thawing causes fat decomposition in milk, which reduces its nutritional benefits. This indicated the need for a flexible pasteurization system that could process lower volumes of milk at a time.

We asked stakeholders about their confidence-level with the end product of flash heat pasteurization. Technicians indicated that they were careful when performing the procedure and made sure that it is done correctly. They were confident that the pasteurized milk was safe to be fed to babies. However, they said that the water-bath reaching a rolling boil was not a clear indicator of when to remove milk from the bath, so typically they waited for a short period of time after water started to boil and then removed the milk. Supervisors had a similar concern with the water-bath's rolling boil being used as a visual indicator to remove milk. They wanted a more definitive indicator to ensure that milk was heated appropriately. This corroborated our in-lab experiments discussed in Section 5.4.1.

We also asked if they wanted to see changes or improvements in the existing flash heat pasteurization process. Two technicians were concerned about contamination of milk during pasteurization. The milk jar did not have a lid in this procedure (see Figure 41), so the technicians expressed concern that they had to be extra cautious to prevent anything from falling into the milk while it was being pasteurized. Supervisors shared this concern and were also concerned about post-pasteurization contamination. Milk is transferred to a smaller jar that has a lid for storage after it has been pasteurized. This introduces the risk of post-pasteurization contamination. Further, traditional flash heat pasteurization processes only one milk jar (with 120 mL of milk) at a time, and supervisors expressed the need to increase processing volumes (that being said, they already knew that our new system could pasteurize up to three bottles of milk at a time).

In addition to formal interviews, we had informal conversations with supervisors to understand how they managed information about donors and pasteurization procedures. All data was recorded on paper. Each milk bank maintained a donor registry that had information to identify each donor and a timeline of when milk was collected from them. There was a separate registry for recording pasteurization procedures. After pasteurization, milk bottles were labeled with the donor's ID, a batch ID, and the date on which the pasteurization was performed. This system worked well as milk banks operated in silos; however, during the deployment period, we observed that pasteurized milk bottles were exchanged between milk banks. This helped us

realize that as milk banking scaled up in South Africa, the exchange of milk between milk banks would increase, and staff would benefit from a digital information management system.

To collect baseline temperature data for flash heat pasteurization, we had a technician and a supervisor perform two runs of milk pasteurization using the traditional method (we had a temperature probe immersed in milk to passively record temperatures). We asked them to perform the procedure as they normally would and told them that we needed this data only for our research. We found inconsistencies in the peak temperatures achieved by milk, specifically; it never even reached a temperature of 70$^{o}$C. This raised concern amongst supervisors and highlighted the risk with the subjective interpretation of using the rolling boil of the water-bath as a visual indicator to stop heating milk. The procedures took close to an hour in all the runs. While the water-bath started boiling fairly quickly, usually within ten minutes, cooling took a longer time. These experiments in the field validated the results from our in-lab experiments, which indicated that real-time temperature monitoring and clear feedback would enable better control of the pasteurization process (cf. Section 5.4.1).

### 5.5.2 System Enhancements

We set up our system in the pediatrics lab at UKZN in parallel with visiting milk banks and collecting baseline data. After completing the baseline interviews with the supervisors, we demonstrated our system to them to get feedback and incorporate any changes. Looking at graphs from a few procedures on the server's web interface, they noticed that the cooling phase of the process was taking too long. They explained that in order to minimize nutritional and immunological degradation during pasteurization, it is important to cool milk as quickly as possible after the high temperature threshold is reached. They helped us devise a more efficient cooling method that shortened the procedure time to about 17 minutes. With this mechanism, after the high temperature threshold is reached, the jar holder is removed from the hot water-bath and placed in a water basin at room temperature. After a minute, ice packs are added to the water basin to speed up the cooling process. This gentle transition from hot water to water at room temperature to ice cold water ensures that jars do not break but that milk cools much faster. We modified the mobile application to guide users through this 2-step cooling process. Figure 56 shows step 2 of the cooling process along with the corresponding screenshot of the mobile app.


**Figure 56: The 2nd step of the 2-step cooling process. The inset is a screenshot of the mobile app, which prompts the user to add ice packs to the water basin. Photo credit: King Edward Hospital, Durban, South Africa.**

During the baseline data collection, we learned about the information management needs in human milk banking. It was not possible to implement a complete information management system during the short deployment period. However, we made a start in preparation for scale-up. We added a new screen to the mobile application to enter donor information (shown in Figure 57); this information also gets uploaded to the server.


**Figure 57: Pasteurization monitor V2 at the Neonate ICU of King Edward Hospital. The milk bank technician is entering donor information in the mobile app. The inset is a screenshot of the Activity that lets the technician enter information. Photo credit: King Edward Hospital, Durban, South Africa.**

In addition to archiving and tracking pasteurization procedures on the server, we saw an opportunity to provide tracking at milk bank facilities as well. We integrated a Bluetooth-enabled printer with our mobile application, letting technicians print pasteurization reports and labels for pasteurized milk jars. Figure 58 shows a screenshot of the Android Activity used for printing. Figure 59 shows printouts of a bottle label and a pasteurization report. The pasteurization report summarizes the procedure and includes the date, batch ID, donor IDs and temperatures achieved during pasteurization. The date and IDs are also encoded in a linear (one-D) barcode that is printed on the report. The labels for milk jars have the date, batch ID, donor ID and an expiration date. Like the pasteurization report, labels also have the date, batch ID and donor ID encoded in a one-D barcode that gets printed on each label. The barcodes also include the mobile phone's IMEI to uniquely identify the facility where the procedure was performed. As human milk banking scales up in South Africa and pasteurized milk jars are transferred between facilities, we expect that these barcodes will provide the capability to uniquely identify and track individual jars.

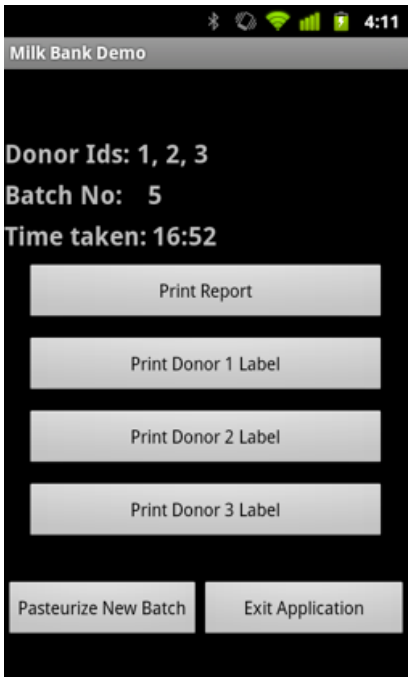



**Figure 58: A screenshot of the Android Activity that allows technicians to print pasteurization reports and labels for pasteurized milk jars at the end of each procedure. The mobile app uses a Bluetooth-enabled printer to accomplish this.**

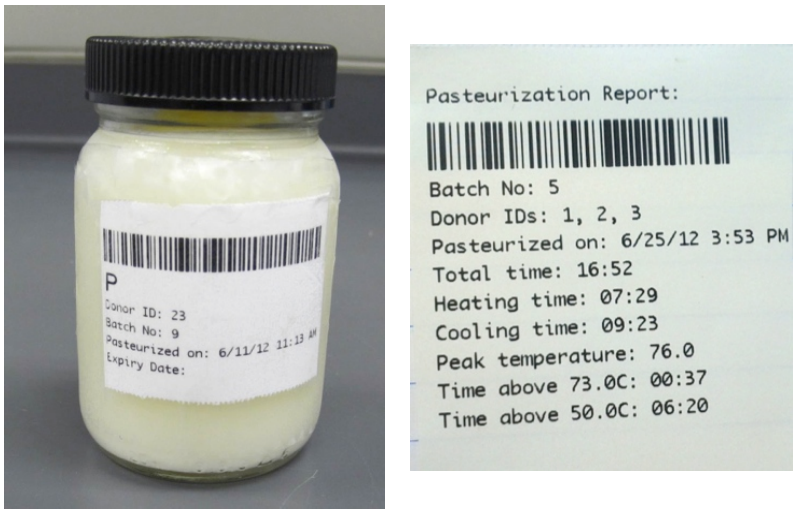



**Figure 59: (Left) A pasteurized milk jar with a label. (Right) A pasteurization report summarizing a procedure. The report becomes a part of the registry maintained by technicians at milk banks. Print outs have a linear barcode that uniquely identify a milk jar or a procedure. Photo credit: Pediatrics Lab, University of KwaZulu-Natal, Durban, South Africa.**

The complete pasteurization process, including the enhancements we made in Durban, has the following steps:

1. The user places the stand with the water and milk jars in the water-bath and starts the mobile application.
2. The user enters donor information in the mobile app and starts heating the water-bath.
3. The mobile app monitors temperatures and guides the user through the process.
4. Audiovisual feedback alerts the user before the high temperature threshold is reached.
5. Feedback intensifies at the high temperature threshold until milk is removed from the water-bath.
6. The user cools the milk using the 2-step cooling process.
7. When milk cools down to the low temperature threshold, the user gets feedback that the process is complete.
8. The mobile app lets the user print a pasteurization report and labels for pasteurized milk bottles.
9. The temperature-time data from the pasteurization and donor information is transmitted wirelessly to the central server for archiving and review by supervisors.
10. Supervisors access the server's web portal to generate reports and perform audits.

A video demonstration of our system is available at [85].

### 5.5.3 User Training

We conducted training sessions with users to show them how to use and maintain the system. The user group included everyone from the baseline interviewee group except the neonatologist. We conducted three training sessions over three days; in each session first we demonstrated system setup, which included measuring the appropriate volumes of liquid for the glass jars, water-bath and cooling water basin. We showed how the mobile phone is connected to the USB Bridge and how the Android application is started and used. We also covered common problems that could occur (e.g., the mobile application not being able to communicate with the temperature probe) and trouble shooting steps. Second, we demonstrated the entire pasteurization procedure. Third, we had each user set up the system and perform a pasteurization procedure. Most users were unfamiliar with touch screen phones; however, they got comfortable with the Android application's user interface by the end of the second training session. By the end of the third training session users could perform procedures without any assistance from us. Finally, after the third session, we let the users tinker with the system independently while we were available to answer any questions.

Figure 60 shows a training session where milk bank technicians worked with our system. We also created a user manual and a training video to help with scale-up efforts.

**Figure 60: A user training session in the pediatrics lab at the UKZN. The three participants are technicians at milk banks in the Durban area. Photo credit: Pediatrics Lab, University of KwaZulu-Natal, Durban, South Africa.**

We recruited local engineers to help maintain the system. This has been one of the key reasons for the success of the project so far, and their support will be critical for the long-term sustainability and scalability of the system. During our time in Durban, the engineers enhanced some aspects of the system and continued to do so after we left. They improved the stand and also identified a local engineering firm to get it manufactured locally. Before leaving Durban, we also installed our system at the NICU in KEH, where it is used to do routine pasteurizations.

### 5.5.4 Scale-Up

Section 5.3.2 identified some issues in the second version of the pasteurization monitor that was deployed in Durban over the summer of 2012. After returning from that trip we addressed these issues and created the next version (cf. Section 5.3.3). We returned to South Africa in November 2012 with the new system and upgraded existing installations. Additionally, we installed the system at three new NICUs. Two of the locations were in Durban while the third was in Paarl, Cape Town. We are now working to scale up the system further and establish new milk banks that will use our system. Since the summer of 2012, word of our system has spread within the milk banking community, and we have been getting requests from organizations in a few developing countries (Kenya, Ethiopia, Vietnam and India) who want to expand human milk banking and want to purchase our system. We are working to address these needs and make this technology intervention sustainable.

Our mobile application still uploaded data to the server over a cellular connection that required a data plan. This was a barrier, especially since the data does not need to be uploaded in real-time; for this application, it is acceptable to upload data in bulk and periodically. During this second trip, we realized that most of the supervisors had convenient access to WiFi networks, either at home or at public places like coffee shops. So we enhanced our mobile application and implemented an option that lets users upload historical pasteurization data over a WiFi connection. Data can still be uploaded in real-time if the mobile phone is provisioned with a data plan; otherwise, it is just stored on the device, for uploading later.

### 5.5.5 Results

To date, our system has been installed at five locations in South Africa, and scale-up efforts are in progress. Our colleagues in the pediatrics lab at the UKZN used the system to pasteurize donor milk and perform microbial activity analysis on pre- and post-pasteurized

samples. They processed a total of 100 samples (obtained from different donor mothers) of which 82 samples showed microbial contamination before pasteurization; however, no sample showed microbial activity post-pasteurization. This validated the safety of the procedure in lab settings. Temperature graphs accessible from the server show that the procedures are consistent in terms of peak temperatures achieved, the duration for which temperature is above 55$^{o}$C (research indicates that pathogens become denatured above this temperature [86]), and the total time taken to complete procedures. Improved quality control as indicated by these results has boosted the confidence of the staff performing the procedures. Supervisors have said that the sophistication added by the new system, especially barcoded labels on pasteurized milk jars, enhances the satisfaction of families who need donor milk for their babies by reinforcing the perception that they are, in fact, receiving a medical-grade product.

Our system makes several improvements to the traditional flash heat pasteurization process. Continuous temperature monitoring, with clear audiovisual indicators for each stage of pasteurization, is very important for producing repeatable, consistent results. Our system reduces the chances of contamination during or after pasteurization by monitoring the temperature to control pasteurization and placing milk in sealed jars with lids. Barcoded labels printed by the mobile application uniquely identify pasteurized milk jars and enable them to be tracked. Finally, we reduced the procedure time by a factor of three and increased the volume of milk processed per procedure by another factor of three.

To date, a total of about 1000 pasteurization procedures (one to three milk bottles per procedure) have been done in South Africa using our system. One of the milk banks reported contamination in seven post-pasteurized samples. The temperature curves for these procedures look normal, indicating that the pasteurization was done correctly. Our colleagues with a background in microbiology confirmed that the pathogens found in the contaminated samples were not heat-resistant, so they should have been destroyed during pasteurization. They say that post-pasteurization contamination is the most likely reason for this. This highlights the need for strict guidelines to ensure sterility, and always performing microbial analysis on post-pasteurized milk samples before feeding to babies.

We learned of operational problems at the facilities in the months following our second trip to South Africa 2012. A technician at one milk bank reported that the printer had stopped working; the printer's battery was dead and just had to be recharged. A couple of milk banks

were slow in getting started because they did not have the appropriate staff to perform routine activities, like recruiting donor mothers from NICUs and pasteurizing donor milk. Even though it is a national priority in South Africa, the Department of Health has not yet finalized their guidelines for human milk banking. So it has been difficult to allocate resources for human milk banks at the hospitals. We expect that such issues will be resolved over time as the government rolls out and starts implementing the guidelines. Support from the government is critical for the long-term sustainability of this project.

## 5.6 Related Work

The work presented in this chapter is an example of a mobile cyber-physical system (CPS), defined in [87] as a computer system that processes and reacts to data from external stimuli from the physical world and makes decisions that also impact the physical world. White et al. discuss some R&D challenges inherent in mobile CPS in [88]. They identify interfacing with external sensors as one of the challenges and propose to leverage a service-oriented device architecture [89] and the Device Profile for Web Services [90] to address it. Resource-constrained sensor platforms become part of IP-based cyber-physical systems by implementing scaled-down web services protocols. Our work leverages ODK Sensors (cf. Chapter 3) that makes it easier for mobile applications to interface with external sensors over wired or wireless communication channels. Our system makes sensor data available to IP-based cyber-physical systems without requiring web services protocols to be implemented on sensor platforms.

Commercial pasteurizers, like the one shown in Figure 40 [91], do provide temperature monitoring and archival capabilities. However, they require a temperature probe to be connected to a co-located computer, which may not be practical for low-resource environments. Additionally, it is difficult to move the archived temperature data from the computer on which it is recorded to a central database; this requirement will need to be addressed as milk banking scales up in countries. Our system is based on mobile phones, which lowers the infrastructure requirements, and the ability to move temperature data is already built into the mobile application.

## 5.7 Conclusion

We have developed a mobile sensing system to manage donor breast milk pasteurization at human milk banks. Our system lets milk banks be administered in a decentralized manner. It

enhances flash heat pasteurization by adding temperature monitoring and archiving as well as audiovisual feedback to guide milk bank technicians performing the procedure. Continuous temperature monitoring and feedback helps ensure that pasteurization procedures are performed correctly. Archiving temperature data on a server accessible over a web browser lets supervisors remotely monitor procedures and ensure procedural compliance. Additionally, the system enables technicians to print pasteurization reports and labels for pasteurized milk jars via a Bluetooth-enabled printer. The printouts have linear barcodes that encode information to uniquely identify individual milk jars. This capability will prove useful as human milk banking is scaled up and milk jars are transferred between facilities. The total equipment-cost of our system is under $600 USD.

In collaboration with PATH and our in-country partners at the HMBASA and the UKZN, we deployed our system at five locations in South Africa. Four of these deployments are at the milk bank in neonate wards of hospitals, where the system is used to perform routine pasteurization of donor breast milk. The other deployment is at a pediatrics lab in the Nelson Mandela School of Medicine at the UKZN. Researchers in the lab used our system to pasteurize breast milk and validate its safety. They processed microbial assays on 100 samples of donor milk; 82 of these samples showed microbial activity before pasteurization, while there was no microbial activity in samples taken after pasteurization. PATH and HMBASA are now working with the South African Department of Health to implement the system at milk banks across the country.

The deployments in South Africa gave us a better understanding of the human milk banking ecosystem. We identified the need for a unified information management system to address the information needs of the entire milk banking process, from recruiting and screening donor mothers to feeding pasteurized milk to infants. We believe that such an information system will play a critical role in the adoption of decentralized human milk banking in countries. Our work to date is a start at creating information management tools that will help governments implement low-cost, decentralized human milk banking, and we intend to continue developing it further.

Microbial analysis of post-pasteurized milk is an essential part of the human milk banking process. It is currently a bottleneck because milk banks send samples to central labs; if the labs are backed-up, the turn-around time could be on the order of weeks. In collaboration

with PATH, we will develop a simple diagnostic test that can be done at the milk bank itself to verify if the milk is safe to be fed to babies. We expect this to further simplify decentralized human milk banking.

Since the summer of 2012, when we did our first field deployment, word of our system has spread within the milk banking community, and we have been getting requests from organizations in some developing countries who want to expand human milk banking and want to purchase our system. We are working to make this technology intervention sustainable and scale it up to meet the global needs.

# Chapter 6

# Measuring Time-use

Access to clean and potable water, critical to health, is not always available in rural areas, causing some residents to venture far from home to collect water daily. This collection burden disproportionately falls on women and children; however, little research quantifies the amount of time spent doing this chore. (This chapter uses the term "time-use" to refer to the time spent daily for collecting water). The work presented in this chapter seeks to quantify water collection efforts in order to understand the socioeconomic impact of water policies and water source locations. Our technical contribution enables researchers to gather accurate and individual time-use data easily and efficiently.

To enable these measurements, we built a low-power wireless system that uses Bluetooth-enabled smartphones to retrieve data from sensors attached to household water containers and to upload the data to a computer. While Chapters 4 and 5 covered applications that processed data from sensors in real-time, this work represents a class of applications that records sensor data in the field over an extended period of time and periodically retrieves it using an aggregator device. Thus, this chapter supports contribution 3 of the thesis: **Implementation and evaluation of sensing systems to enhance workflows in low-resource settings.**

The chapter begins by describing the need for accurate time-use measurement in rural communities. Section 6.2 provides an overview of our sensing and data collection system, while Section 6.3 relates how researchers use the system to collect time-use data. Section 6.4 presents a field trial of the system done in rural Ethiopia in 2011 and is followed in Section 6.5 by a discussion of this work and how it can be adapted to other domains. We position our work in the context of related work in Section 6.6 before concluding in Section 6.7.

## 6.1 Motivation

Improving access to safe, conveniently located drinking water remains a key development challenge in many parts of the world. According to the WHO and UNICEF Joint Monitoring Programme (2010) [92], approximately 884 million people worldwide, 37% of whom live in sub-Saharan Africa, still use unimproved drinking water sources, like rivers or springs. Among policymakers and researchers in the water sector, the predominant focus has been on the health benefits of improving access to safe water, particularly in preventing diarrheal disease [93]. While collecting water from unimproved sources can take up to six hours in some cases, the establishment of community or household water taps can significantly reduce time-use. A number of studies based on household surveys have found that water project beneficiaries often perceive the increased convenience and time savings as equally or more important than health benefits [94]. In [95] Cairncross and Valdmanis observe that "given the relevance of the time-saving benefit to water supply policy, and the fact that the benefit is usually uppermost in the mind of the consumer, it is remarkable how few data have been collected on the amounts of time spent collecting water".

What impact does providing a convenient water supply have on water carriers' patterns of time-use, and what is that saved time used for? One difficulty in answering this question is measuring time-use accurately. A commonly used approach asks respondents directly: "how many minutes did you spend yesterday collecting water?" This question suffers from recall bias and also presumes a familiarity with thinking about time in terms of hours and minutes. To minimize recall bias, researchers ask subjects in industrialized countries to complete time-use diaries, recording their primary (and sometimes secondary) activities over one day or more; this has been impossible, however, for illiterate populations in poor countries. Further, at the project's site in rural Ethiopia, only 53% of the population owns some type of time-keeping device (a watch, clock or mobile phone).

In an earlier study, members of our team (researchers in the Evans School of Public Affairs at the University of Washington) elicited time-use data before and after water supply projects were completed in three villages in rural north-central Ethiopia. Time was measured in three ways: via direct recall, through an interviewer-guided activity that allocated the total amount of time in the day to different categories, and with a self-completed pictorial diary paired

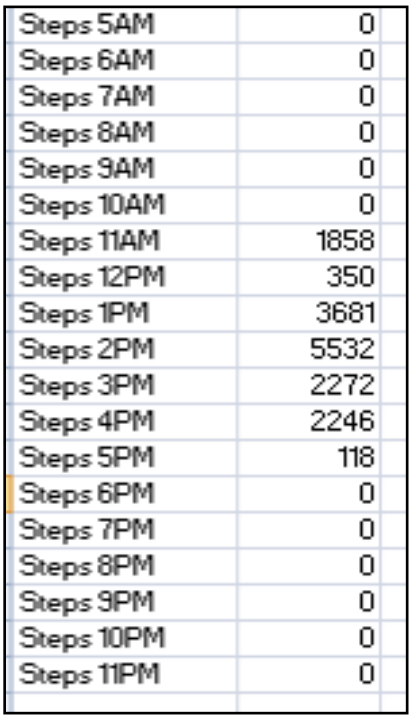
with a simple timer set to beep every 30 minutes. However, it proved difficult to determine which of these methods was most accurate without ground-truth validation of time-use.

**Figure 61: Water vessels (the yellow-colored containers, aka Jerry cans) commonly used in rural Ethiopia to collect water. Photo credit: Water1st International, Seattle, USA.**

As a first step toward identifying accurate alternatives to measure time-use, our Public Affairs colleagues conducted a pilot study in the summer of 2010; for the pilot, Omron HJ720ITC pedometers [96] were attached to containers used for water collection (like those shown in Figure 61). Since pedometers are fairly accurate in counting footfalls, the intention was that this approach would give an accurate measure of time-use. The HJ720ITC reports step-counts at a minimum granularity of 1 hour and can upload data to a computer over a USB connection. Figure 62 shows the hourly step-counts from the pedometer used by one of the participants. In the post-study interview, the participant reported that she spent 90 minutes after 11AM collecting water on that particular day. However, the pedometer recorded steps in the 11AM – 6PM time period because the water container was used for household activities after water collection.

It was difficult to determine time-use based only on the pedometer's readings because step-counts were reported at an hour's granularity. For instance, steps recorded when the

container was used for shorter duration activities (e.g., washing hands) could not be eliminated as noise in the reported data. Step-count logs could be retrieved from pedometers over a USB connection with a computer; however, due to the rough commutes to and from the villages, it was difficult for researchers to carry a laptop for data collection. Finally, scaling up deployments based on these pedometers would have been difficult due to its high price ($60 USD per unit, in 2010). Due to these limitations, the pedometer-based approach for measuring time-use was not pursued further.



| | |
|---|---|
| Steps 5AM | 0 |
| Steps 6AM | 0 |
| Steps 7AM | 0 |
| Steps 8AM | 0 |
| Steps 9AM | 0 |
| Steps 10AM | 0 |
| Steps 11AM | 1858 |
| Steps 12PM | 350 |
| Steps 1PM | 3681 |
| Steps 2PM | 5532 |
| Steps 3PM | 2272 |
| Steps 4PM | 2246 |
| Steps 5PM | 118 |
| Steps 6PM | 0 |
| Steps 7PM | 0 |
| Steps 8PM | 0 |
| Steps 9PM | 0 |
| Steps 10PM | 0 |
| Steps 11PM | 0 |

**Figure 62: Pedometer data for a participant who reported collecting water for 90 minutes after 11am. Image credit: Prof. Joe Cook, UW, Seattle, USA.**

## 6.2 System Overview

Shortcomings of the previous approaches led us to develop a better solution: a sensor for measuring time-use that addressed limitations identified in the pedometer trial. As done in that trial, our sensor was also attached to containers used for water collection and measured time-use by monitoring motion. However, it could identify movements of shorter duration, thus making it possible to suppress noise in the signal stream. Since the sensor was Bluetooth-enabled, smartphones could retrieve data over a wireless connection. The phones, in turn, could upload this data to a central database for easy access by researchers. Even though Bluetooth consumes

significant power, we added it to the board to lower a data collection barrier specific to developing regions: Time-use researchers spend a lot of time travelling between villages to collect data, and carrying a laptop (or even a netbook) for collecting data from sensors, possibly over USB, is inconvenient. Using Bluetooth-enabled sensors lowers this barrier significantly by enabling mobile phones to be used for data collection. We found this convenience/energy tradeoff to be acceptable because the Bluetooth radio is turned on only when the researcher and phone are co-located with the sensor, and then only for long enough to exchange the relevant data.

Our end-to-end system has three main components: (1) the sensing subsystem, (2) an Android smartphone-based data retrieval and configuration management tool, and (3) a deployment data aggregator tool to collect data from smartphones. The design and implementation of these components is discussed below.

### 6.2.1 Sensor Subsystem

We chose the Atmel processor-based Arduino microcontroller for our sensor because it is competitive in its energy efficiency compared to other microcontrollers and is easy to program. Even though low-power accelerometers are typically used for motion tracking applications, we used small mercury tilt switches [23] that do not require any power for operation. Mercury switches can be used as binary indicators of motion (1 when motion is detected, 0 otherwise), which was sufficient for our purposes. Time-use trips are defined by a start-time and an end-time (4 bytes each). During testing, we verified that two switches placed orthogonally to each other are sensitive enough to track motion for our application.

Our application required accurate timestamps to record time-use; since the Arduino does not have an internal real-time clock (RTC), we integrated an external RTC module on the board. An external 32KB EEPROM chip was also added to persistently store up to 2 months of time-use data. A Bluetooth module enabled communications with a Bluetooth-enabled mobile phone. Two AA Alkaline batteries provided 2700 milli-amp hours of power to the system at 3 volts. These batteries were chosen for their ubiquity even in the developing world. The battery choice might change in the future (to rechargeable batteries), but it is straightforward and fast to simply replace the batteries at the end of each data collection rather than recharging them in the field, which at present takes too much of the data collectors' time.

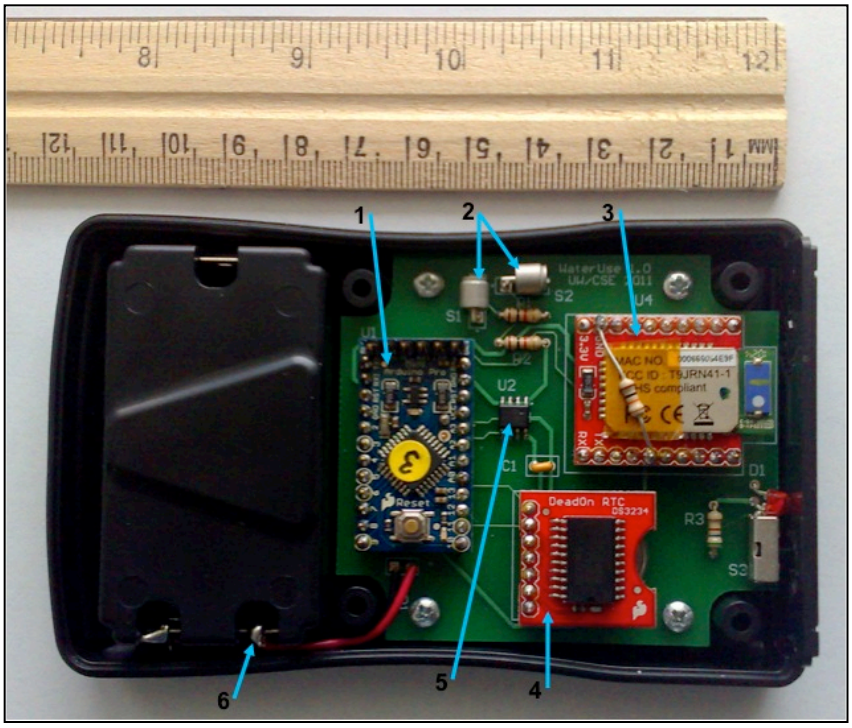



**Figure 63: Motion sensor used to record time-use. Its components are: (1) Arduino microcontroller, (2) Mercury switches, (3) Bluetooth module, (4) RTC module, (5) EEPROM chip, and (6) 2 AA batteries (not visible on the left). The ensemble is packaged in a black box (whose final dimensions are 4.5" x 2.5" x 1").**

Our motion-sensing device is shown in Figure 63. Due to the prototype nature of our hardware, its current cost is on the higher end, but when produced at scale we expect the cost to drop to under $20 USD per unit.

#### *6.2.1.1 Power Management and Data Transfer*

We learned from our Public Affairs colleagues that water collection trips could last up to six hours. To maximize the system's battery-life, we programmed it to remain in its low-power state at all times. It is woken up only by a change in state of the mercury switches.

Bluetooth has the highest power budget amongst all components, so it is always powered off to conserve energy. Researchers turn it on to configure the sensor before deploying at a household or to retrieve data (both of these operations are discussed in more detail in Section 6.3). The switch to control power to the Bluetooth module is hidden inside the enclosure so that household members do not turn it on accidentally while using the water container. After turning on Bluetooth, researchers must shake the sensor so that interrupts from mercury switches wake

the Arduino up, allowing it to accept and process commands sent in from researchers' phones over a Bluetooth connection.

```
/* variables initialized on device startup:

maxIdleDuration
minMovementDuration
EEPROMwriteIndex
CTS ← time from RTC
STS ← CTS
ETS ← CTS
*/
if BT_ON & BT_SERIAL_CONNECTED:
     //connected to mobile phone
     //receive configuration OR
     //upload trip data from EEPROM

else:
     //trip detection algorithm
     CTS ← time from RTC
     if CTS-ETS > maxIdleDuration:
          //start of new trip
          if ETS-STS > minMovementDuration:
               //increment EEPROM pointer
               EEPROMwriteIndex++
          STS ← CTS
     else:
          if ETS-STS > minMovementDuration:
               //update EEPROM
               EEPROM ← (STS, ETS)
     ETS ← CTS
     enter_low_power_mode()
```

**Figure 64: Sensor's main-loop to communicate with phones over Bluetooth and record time-use data.**

The sensor's main-loop is shown in Figure 64. Each iteration handles possible communication with a connected mobile phone or, more typically, executes the trip detection algorithm described below. The trip detection algorithm records the time and puts the system back to sleep to await the next bounce in a mercury switch.

#### *6.2.1.2 Trip Detection Algorithm*

Since water collection trips are always at least a few minutes long, we only record trips that are longer than *minMovementDuration* minutes. An ongoing trip is considered complete if no movement is detected for *maxIdleDuration* minutes. The algorithm begins by querying the

current timestamp (*CTS*) from the RTC. If it determines that a new trip has started, the pointer to the EEPROM's next free location is incremented (*EEPROMwriteIndex*) so that the new trip's timestamps can be recorded, and the trip's start timestamp (*STS*) is updated. Otherwise, if the movement is already part of a valid trip (i.e., greater than *minMovementDuration*), then the trip's start and end timestamps (*STS* and *ETS*) are updated in the EEPROM. Finally, the algorithm update's the trip's end timestamp (*ETS*) if no movement is detected for *maxIdleDuration* minutes and terminates by transitioning the device to its low-power mode. We expect the device to be in its low-power mode (i.e., stationary) for most of the day, transitioning into active mode only when motion is detected. Bluetooth will be active at least two times during each device's deployment (initially to configure the device and at the end to retrieve data).

#### *6.2.1.3 Battery Life Projections*

We estimate the device's battery-life based on the following usage profile. The Arduino draws about 4.5 milli-amps of current in active state and under 20 micro-amps in its low-power state. The Bluetooth module connected to the Arduino draws about 20 milli-amps on average when in use. As a conservative estimate, if we assume the container to be in motion for about 6 hours per day and Bluetooth to be turned on for 10 minutes every day, we estimate a battery life of over two months. Figure 65 shows the projected battery life of the system as a function of the percentage of time the system spends in its active state. The red marker on the graph is our conservative estimate of battery life.

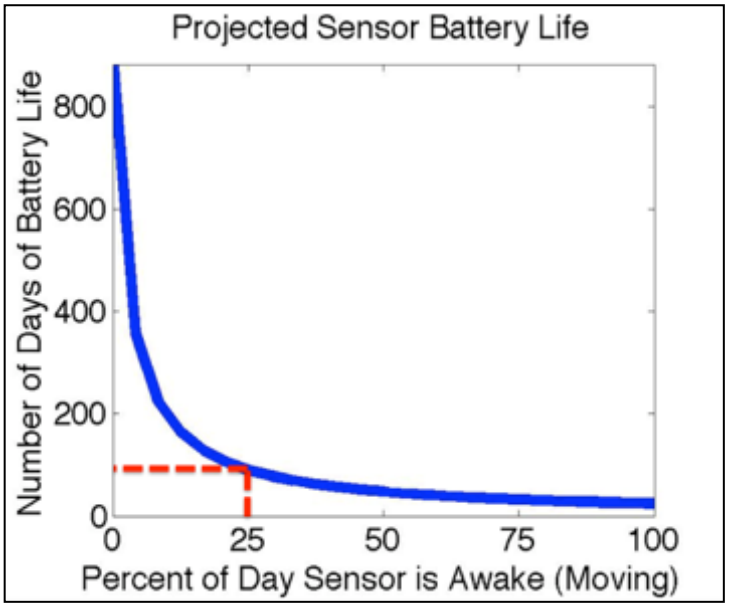



**Figure 65: Battery life vs. time spent collecting water per day.**

### 6.2.2 Android Application

Android smartphones play an important role in the system by acting as intermediary mobile nodes that manage the assignment of sensors to households and the collection of data from them. Using mobile phones as intermediaries makes time-use data collection and reporting much easier because researchers do not need to carry laptops or netbooks when they travel to their deployment sites. Assignment of a sensor to a household involves collecting some information about the household and configuring the sensor (as discussed in the next section). The Android application uses Open Data Kit (ODK) [1], an open-source data collection toolkit, to collect the required household information. Sensor configuration is done by the application over Bluetooth. The Figure 66 screenshot shows the top-level operations (i.e., the main activities) performed by the application. The Figure 67 screenshot shows the Activity used to retrieve trip data from a sensor.

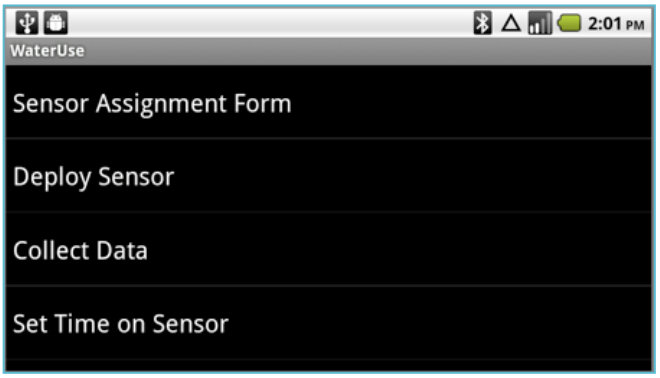



**Figure 66: Main Activity of the Android application that interacts with sensors. "Sensor Assignment Form" invokes ODK to collect information about the household. "Deploy Sensor" programs the unique ID of the household into the sensor. "Collect Data" retrieves data from the sensor. "Set Time on Sensor" update's the time of the sensor's RTC module.**

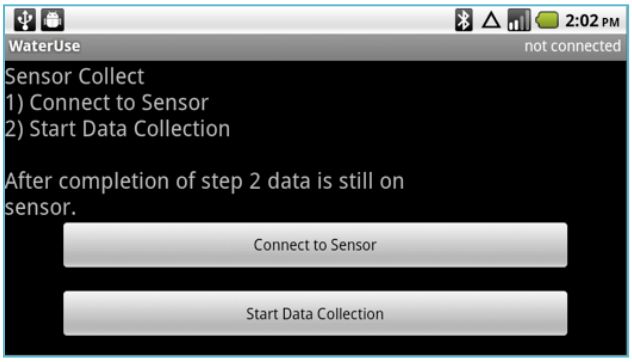



**Figure 67: The Activity to retrieve trip data from the sensor. After connecting to the sensor over Bluetooth, researchers choose the "Start Data Collection" option to retrieve data.**

### 6.2.3 Deployment Data Aggregator

Information from mobile phones used in the field for sensor deployments and data collection is aggregated into a central database. This database could exist in the cloud; however, since reliable Internet connectivity cannot be assumed during these deployments, the database currently exists on researchers' personal computers (typically laptops). A Python program running on the aggregator machine receives all the deployment data from mobile phones and stores it in a local SQLite database. Because Android phones also have SQLite, the central database could also be pushed out to phones for use in the field.

The aggregator lets researchers import sensor-household assignments done in the field as well as time-use data collected from sensors. Data stored in the local database can be exported to comma-separated-value (CSV) files to facilitate analysis using a program like Microsoft Excel or imported into relational databases and statistical packages.

## 6.3 Time-use Data Collection

Time-use study of households using our system happens in three phases. In Phase 1, a sensor is deployed at a household (i.e., attached to a water container in the house). At this time the sensor is considered to be assigned to that household for the purposes of the study. Additional information about the household collected at the time of deployment is shown in Table 13. This information is collected using ODK Collect, which runs on researchers' Android phones. The Android application also assigns a unique and anonymous ID to the household. This ID is programmed into the sensor over Bluetooth to ensure that any future data collection can be linked to the configuration data. For this interaction, the sensor's Bluetooth radio is turned on by the researcher and then turned off immediately afterwards. This procedure is performed for each household participating in the time-use study. Phase 1 is depicted in Figure 68.

| Data | Description |
|---|---|
| Sensor ID | Unique ID for each sensor |
| House ID | Unique (anonymous) ID for each house in the study |
| Date Deployed | Date of deployment at this house |
| Person ID | Unique (anonymous) ID of person collecting water |
| Can Volume | Volume of the water container in liters |
| Can Picture | Picture of the can |

**Table 13: Data collected in Phase 1 using an ODK form.**

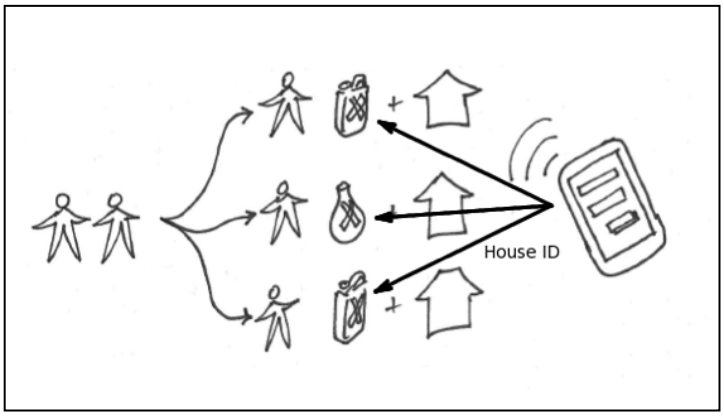



**Figure 68: Phase 1 – Researchers take sensors out into the field for deployment. Sensors are assigned to containers and programmed to store their assigned house ID. Researchers fill out a brief survey to track assignments.**

In Phase 2 (Figure 69), the sensor autonomously detects and records time-use information using the trip detection algorithm described previously. Table 14 shows the data persistently stored by each sensor. The 4-byte SensorID and 4-byte HouseID are stored in the first 8 bytes of the external EEPROM. This is followed by 4-byte timestamps for each trip the sensor detects. At the end of the study-period, researchers return to households to retrieve the sensor data. They turn on the Bluetooth radio of each sensor, activate it by shaking it, and connect to it using their Bluetooth-enabled Android phone. The Android application retrieves the stored data from the sensor and stores it in its local database. The sensor's batteries are replaced depending on the length of the study period. At this point, the sensor can be redeployed to the same household or to a different household if needed.

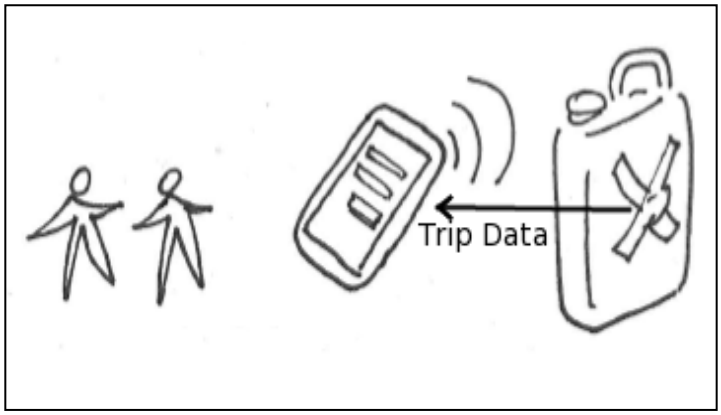



**Figure 69: Phase 2 – Sensors autonomously record time-use information during the study period. At the end of the study period researchers return to collect trip data, replace batteries and redeploy sensors to a different household if needed.**

| Data | Description |
|---|---|
| Sensor ID | Unique ID for each sensor |
| House ID | Unique (anonymous) ID for each house in the study |
| Start Time(s) | The time when a new trip starts |
| End Time(s) | The time when the trip ends |

Table 14: Data stored on a sensor. The SensorID and HouseID are stored as the first 8 bytes of the EEPROM. Each trip is stored in chronological order as a pair of 4-byte timestamps. In Phase 3 (Figure 70), researchers upload all assignment and time-use data from their Android phones to a central computer. The computer's aggregator program merges and stores this in its SQLite database. By the end of phase 3, researchers have all the time-use data in a digital format that can be easily analyzed and shared.

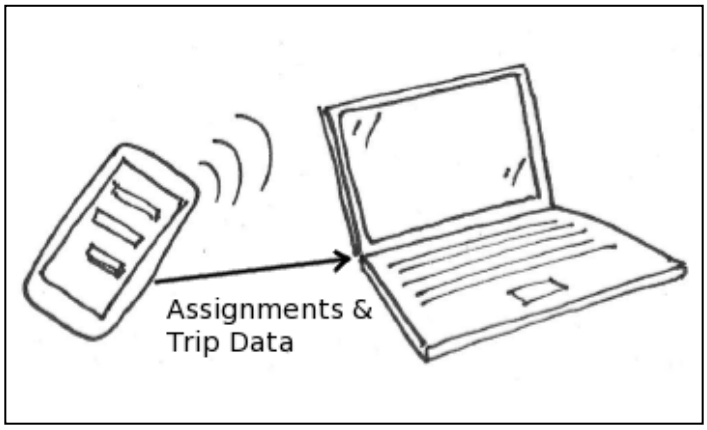



**Figure 70: Phase 3 – Researchers return to their base with Android phones containing sensor data and upload this data from the phone to a computer.**

## 6.4 Deployment in Ethiopia

Development and testing of our time-use monitoring system was completed by May 2011. We then conducted usability trials with the researchers (our collaborators at the Evans School of Public Affairs), who later deployed the system in Ethiopia. After receiving some training from developers, they could comfortably perform all system tasks in the three phases of time-use data collection. Overall, they found the system easy to use and set up.

In the summer of 2011, they travelled to rural Ethiopia to deploy the system under field conditions and to get feedback about sensor usability from the local population. Time-use data was collected from forty households in three villages over a period of four weeks. The villages – Beshikiltu, Kelecho Gerbi, and Tutekunche – are located in the Oromia region (Figure 71) of

Ethiopia. Their water source options include a nearby river, unprotected springs, and a household well. Unless a household had a well, villagers could spend between one to six hours daily collecting water.

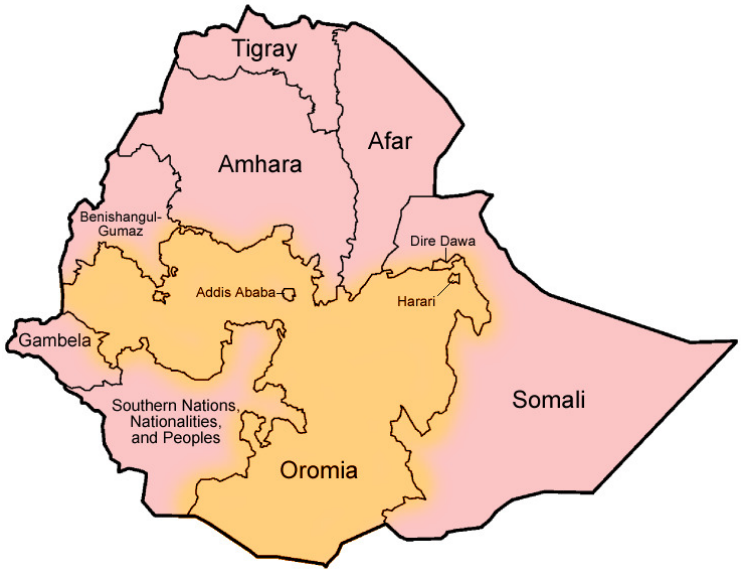



**Figure 71: The Oromia region (colored in orange) of Ethiopia where the system was trialed in the summer of 2011.**

The researchers had four time-use sensors and two Android phones running our application. The sensors were time-shared amongst the households participating in this trial. The *maxIdleDuration* and *minMovementDuration* parameters used in the trip detection algorithm (discussed in Section 6.2.1.1) were set to two minutes in all the sensors. Data collection from each household happened over three days. On Day 1, the researchers deployed sensors at households and collected household information using ODK Collect (Phase 1). Each household typically had a few containers for collecting water; however, only one container per household was used for data collection in this trial. Figure 72 shows a researcher attaching the sensor to a container. As evident from the picture, the sensor mounting and waterproofing mechanism was not very sophisticated, but it was effective nevertheless. On Day 2, a household member used the container to collect water from one of the sources. On Day 3, researchers returned to retrieve the sensor and its data before reassigning it to another household. Sensor data collected on the Android phones was uploaded to the researchers' shared laptop (Phase 3) when they returned to their base at the end of the day.

**Figure 72: Researcher attaching sensor to a water container. The lady is the primary water collector for the household. Photo credit: Yuta Masuda, UW, Seattle, USA.**

Three methods were used to obtain data for ground truth validation. First, researchers conducted a post-study interview with the person who collected water the previous day. This was done to determine if the sensor caused any inconvenience to the water collector and also to get ground truth validation data. Since water collectors do not wear watches, the time of day was reported as one of the following: morning, midday, afternoon or night. The reported duration was their best estimate of the trip's length. The post-study questionnaire is shown in Table 15. Second, water collectors were asked to maintain pictorial diaries to record their daily activities during the study-period. Third, the researchers also obtained GPS data for households and water sources.

Analysis of sensor data from this trial reveals that the recorded water collection trips were less than 2 hours long. 23% of the water collection trips were more than an hour long. Due to budget and time constraints, the researchers could not perform a detailed comparison of data collected from our sensor-based system and the ground truth data obtained from the three

methods mentioned above. However, the remainder of this section compares some results obtained from our system with self-reported data.

| | Questions |
|---|---|
| **1** | What time of day did you collect water yesterday for this trip?<br>(Morning, Midday, Afternoon, Night) |
| **2** | Approximately how long was the trip to the water source? |
| **3** | Approximately how long did you wait at the source? |
| **4** | Approximately how long was your trip back to your home? |
| **5** | Did the sensor make it difficult to carry the jerry can? |
| **6** | Did having the sensor make you uncomfortable? |
| **7** | Did anyone notice the sensor? |

**Table 15: Post-study questions.**

Figures 73 and 74 show trips recorded by one of the sensors deployed at a household. Time periods when the sensor is moving are plotted with a value of 1, while time periods when it is stationary have a value of 0. Thus, the trips detected by the sensor are represented as pulses where the width of the pulse is the length of the trip. Figure 73 shows all trips detected over the 3-day deployment period. Since *minMovementDuration* was set to 2 minutes, the sensor recorded some short duration trips as well. For instance, consider the first two pulses in the graph that occurred on 7/18/11 and lasted for 12 minutes and 2 minutes, respectively. These do not represent water collection trips, but in fact reflect the times when the water container was being used in the house, most likely for activities like cooking or washing. For time-use measurements, such pulses represent noise in the data reported by the sensor and can be eliminated easily based on the width of the pulse.

Figure 73 shows only two pulses long enough to be considered actual water collection trips. These occur back-to-back (a likely candidate for a long round-trip with a short, stationary period in between to fill the container) on 7/19/11: t=11:42 to t=12:51 and t=12:58 to t=14:30 and are highlighted by a red marker in the graph. During the post-study interview, the water collector said that they collected water at midday, and each leg of the trip took 60 minutes. This self-reported data correlates very well with the sensor data. There is a discrepancy of 30 minutes in the duration of the return trip. That could be because self-reported durations are estimates, or because the return trip was slower due to the weight of the water-filled container. Figure 74 shows a close-up view of the water collection round-trip reported by the sensor.

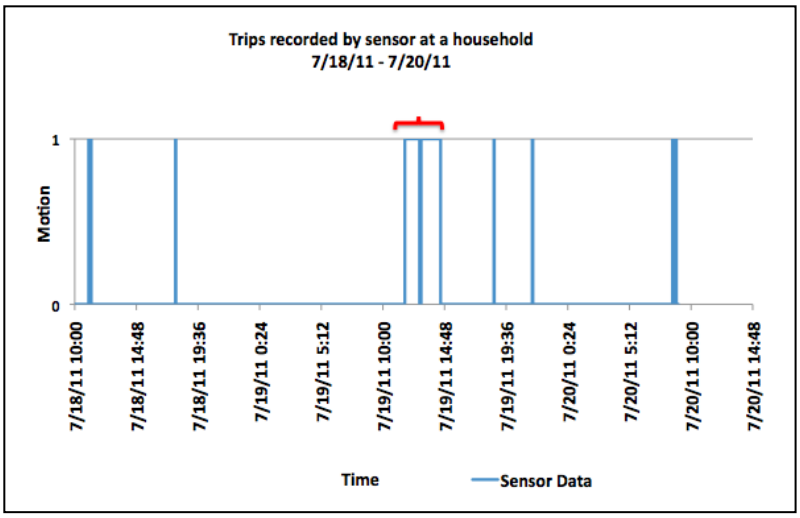



**Figure 73: Trips recorded by a sensor over a 3-day deployment at a household. Trips detected by our sensor are represented as a pulse whose width indicates the length of the trip. Single vertical lines in the graph represent trips of very short duration (possibly household activities in which the container was being used). The 2 pulses with a red marker indicate a water collection round-trip.**

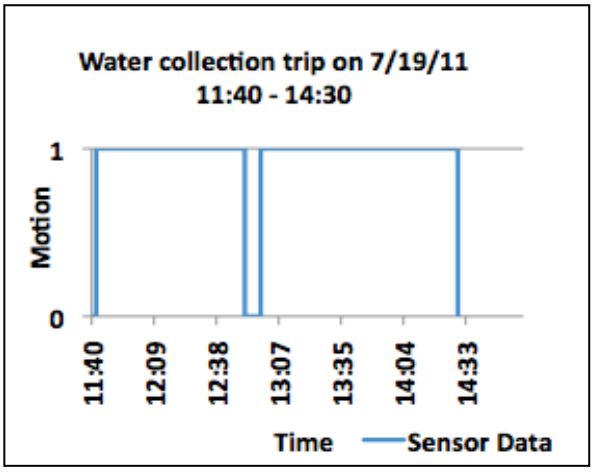



**Figure 74: A zoomed-in view of the water collection round-trip as recorded by the sensor. The length of each trip and time of day correlate well with the water collector's self-reported data.**

Figure 75 shows trips recorded by a sensor at another household over the three day study-period. In this case, the water collection round-trip is recorded in a single pulse (indicated by the red marker) on 7/17/11 from t = 7:30 to t = 9:19. This trip is interesting because only the self-reported time-of-day (morning) correlates with the sensor's recorded data. While the sensor recorded about 110 minutes for the trip, the collector reported spending 3 hours on this trip for water collection (1 hour for each leg of the trip, and 1 hour of wait-time at the water source).

This is an example of a discrepancy between self-reported data and data collected from the sensor.

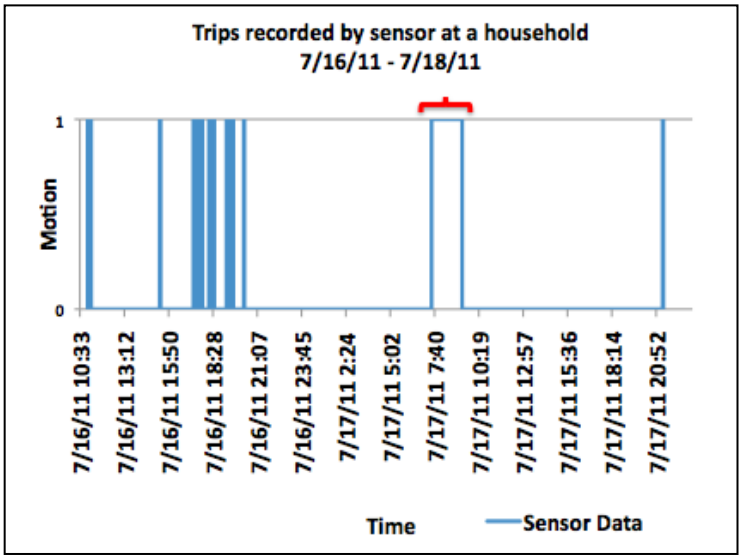



**Figure 75: Trips recorded by a sensor over a 3-day deployment at a household. In this case, 1 pulse (with the red marker) indicates the water collection round-trip. The shorter duration pulses (vertical lines) indicate the water container being moved around, possibly for household activities.**

In addition to getting data for ground truth validation, the post-study interview also tried to gauge the acceptability of the sensor for time-use studies (questions 5 to 8 in Table 15). Of the 40 participants, two used a clay-pot while the others used a jerry can as the water container. The two (5%) participants with clay-pots found it difficult to carry the container due to the attached sensor, the others had no problems. Six (15%) participants were hesitant while carrying their container, fearing that the sensor might break or get detached. Interestingly, 30 (75%) participants reported that passersby asked them about the sensor and what it was being used for. Two participants reported that they had hidden the sensor so that it would not be noticed. Several participants voluntarily expressed a preference for using this new method of data collection over self-reporting via interviews or pictorial diaries. They said that the new method was better for them because it did not require extra time or recall on their part.

Researchers could not get sensor data from some deployments. They reported that the most likely cause for this was that the sensor had stopped working due to either a loose wire connection or dead batteries. Due to limited feedback from the sensor, they realized the anomaly in Phase 2 when they were retrieving data and were unable to connect to the sensor. Despite

these failures, they liked the fact that it was very easy for them to get CSV files from raw sensor data. Moreover, they said that they were easily able to identify noise in the reported data by looking at its graph. The Phase 1 sensor deployment at a household typically took 15 minutes; this included the time needed to attach a sensor to a jerry can. Phase 2 was completed within 5 minutes at each household; this did not include the time needed for the post-study interview. Phase 3 took under a minute to upload data to a computer.

## 6.5 Discussion

The deployment in Ethiopia was the first trial of our system with an end-user community, so we expected to uncover new issues. Nevertheless, the researchers found our system to be useful as it simplified their data collection cycle and got them to the data analysis phase sooner. Although the usability assessment done in Ethiopia was not extensive, the positive response received from study participants and researchers was extremely encouraging, indicating that attaching sensors to containers would generally be acceptable to end-users.

Based on lessons learned from this deployment, we planned some system enhancements prior to a larger-scale deployment. Better feedback mechanisms were needed on the sensor and the Android application to help with quicker diagnosis and rectification of failures similar to those seen in the Ethiopia trial. Packaging of the sensor could be improved to reduce its size and make it more waterproof. This would also have improved usability on containers like clay-pots, which are fragile and have curved surfaces. However, we did not pursue this work further due to budget constraints.

While the system presented here solves a specific data collection problem, it can be generalized to enable data collection in other studies as well. Understanding how people in rural areas spend time more generally is interesting and important to researchers in the Anthropology, Development Economics, Public Health and Social Sciences communities. We believe that appropriate and simple-to-use sensors that provide fine-grained data would be complementary to their existing methods of data collection, which include in-person observations, interviews and standardized surveys.

Our algorithm for detecting time-use trips can be enhanced to monitor other everyday activities – like cooking or washing – that involve water containers. Exploring possibilities to correlate these activities with general health outcomes of households would be valuable. Motion

sensors attached to farming equipment could be used to study time spent in farming activities. Sensors, deployed for days or weeks, and occasionally coupled to smartphones to transfer data, could also be used to monitor water quality and soil or crop status.

Experience gained from our time-use monitoring work is guiding us in the development of another sensor-based data collection application. We are working with researchers at the University of California, Berkeley (UCB), who use DS1922L iButton temperature sensors [97] to monitor cook-stove usage in rural households in Kenya. The DS1922L is a self-contained temperature logger that can be configured to record temperatures in its local memory at different sampling intervals. The recorded data is retrieved via an adapter that plugs into a computer. In their current model, UCB researchers deploy the sensors at households for several days and then return to collect all sensors to retrieve stored data via a computer at a central location.

To simplify cook-stove usage data collection, we modified the DS1922L's adapter to interface it with FoneAstra (cf. Section 2.3). We plan to implement code on FoneAstra so it can retrieve data from the iButton sensor connected to the modified adapter. We will also develop an Android application that will use ODK Sensors (cf. Chapter 3) to retrieve historical sensor data from FoneAstra (to which the DS1922L will already be connected). Thus, in the new model of data collection, rather than bringing all sensors back to a central location, researchers will simply carry a FoneAstra, the modified DS1922L adapter, and an Android phone that can retrieve sensor data from FoneAstra. They will visit households and retrieve data from each sensor deployed in the field. This will save them the round-trip of taking sensors to a central location for data retrieval and then returning them to rural households for the next round of data collection.

## 6.6 Related Work

Architecturally, our system is a three-tier Data MULE (Mobile Ubiquitous LAN Extension, [98]) in which mobile nodes (i.e., smartphones) collect data from sensor nodes and relay it to the central database. In addition to collecting data from sensor nodes, mobile nodes in our system deploy and configure them as well.

Rather than discussing specific works, we refer the reader to [99], which presents a survey of Wireless Sensor Networks in which Mobile Elements (ME) perform data collection. The survey highlights challenges in the discovery phase that are unique to such systems. This is the first step in data collection and enables sensors to discover MEs in the vicinity. Several

protocols have been proposed to perform discovery in an energy-efficient manner, often using secondary, lower power radios. The secondary radio periodically searches for MEs in the vicinity and turns on the primary high-power and high-bandwidth radio for data transfer only when an ME is discovered. In the current implementation, we do not use a secondary, low-power radio in order to maximize the battery-life of our sensor. In our system, researchers explicitly turn on the sensor's Bluetooth radio and shake it to activate the processor prior to data collection.

## 6.7 Conclusion

This chapter presented a system that monitors and reports the time spent by people who collect water (time-use) in rural areas of developing countries. This information can help guide policies that govern how and where convenient water sources should be created in order to increase the positive health-impact on communities. Currently used methods can lead to discrepancies as they rely on self-reporting, which introduces recall bias; this gets amplified because water collectors have a different notion/measure of the passage of time. Our system improves the accuracy of time-use measurement and reporting by attaching low-cost, low-power sensors to containers used for water collection. Bluetooth-enabled Android phones are used to configure the system, deploy sensors at households, and then to retrieve the data recorded on these sensors. This data is then uploaded to a central database on a computer where it can be analyzed or shared further. The system has addressed limitations identified in an earlier study that used pedometers to measure time-use.

We deployed our system in forty households of three villages in the Oromia region of Ethiopia in the summer of 2011. We received positive feedback about the efficacy of the system from field researchers. Several household members who carried containers with sensors attached to them gave positive feedback about the convenience of the system.

# Chapter 7

# Other Applications

This chapter presents other mobile sensing applications that are in various stages of development/trial. These applications also leverage the technologies described in Chapters 2 and 3, demonstrating that the underlying technologies are general purpose and can be used in a wide range of mobile sensing applications. Section 7.1 discusses an application used to teach emergency tracheotomy to users. Section 7.2 describes ongoing work in developing sensor-based tools to assist with the management of cardiovascular diseases. In Section 7.3, we present an application that monitors the gait of people with a prosthetic leg. We then review an application to help with the diagnosis of childhood pneumonia in Section 7.4, after which we conclude the chapter.

## 7.1 Emergency Tracheotomy Training Tool

An emergency tracheotomy (also known as cricothyrotomy) is a rare but life-saving procedure performed by doctors and emergency medical technicians. It is done as a last resort to restore air flow in patients who have an obstruction in their airway and are unable to breath. In this procedure, an incision is made in the patient's trachea (between the thyroid and cricoid cartilages) to insert a tube that restores air flow. Studies indicate that cricothyrotomy procedures performed in pre-hospital settings are often unsuccessful or result in complications [100], [101]. Most unsuccessful attempts are due to inaccurate placement of instruments in the cricoid area or to incorrect identification of the anatomy. However, research from advanced centers that provide extensive training for performing airway procedures, including the use of simulators, shows that pre-hospital cricothyrotomy success rates can be improved significantly with proper training [102]. Such problems are exacerbated in developing countries where personnel to provide training, and training equipment (e.g., simulators) are scarce.

Motivated by this, colleagues in the department of Electrical Engineering (EE) at the University of Washington developed a low-cost cricothyrotomy simulator that uses readily available components and inexpensive sensors. To minimize its cost, they needed a way to interface their simulator to a battery-powered mobile device (instead of a computer or a laptop) so that a mobile application could be developed to guide users while they learn the cricothyrotomy procedure using the simulator. This led to a collaboration in which we integrated the EE simulator with FoneAstra and developed a mobile application that leverages ODK Sensors to get sensor data from the simulator via the FoneAstra board. The application guides the user through the procedure based on the sensor data.

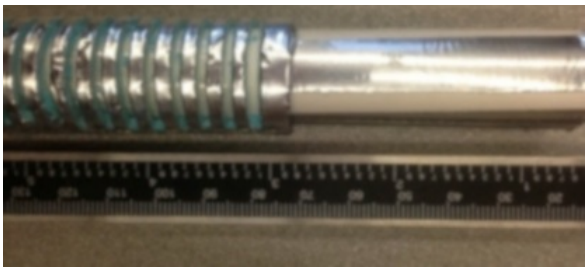


**Figure 76: Disposable tracheal tube model built from thin cardboard with rings of cartilage made from foam strips. Photo credit: UW BioRobotics Lab, Seattle, USA.**

Figure 76 shows the simulator's disposable trachea model, which is replaced after each procedure. It is made of cardboard, and its size is close to the size of an average adult's trachea. Foam strips (0.5"×2", 2 mm thick) are cut to form the cartilaginous tracheal rings and are attached on the trachea with appropriate spacing.

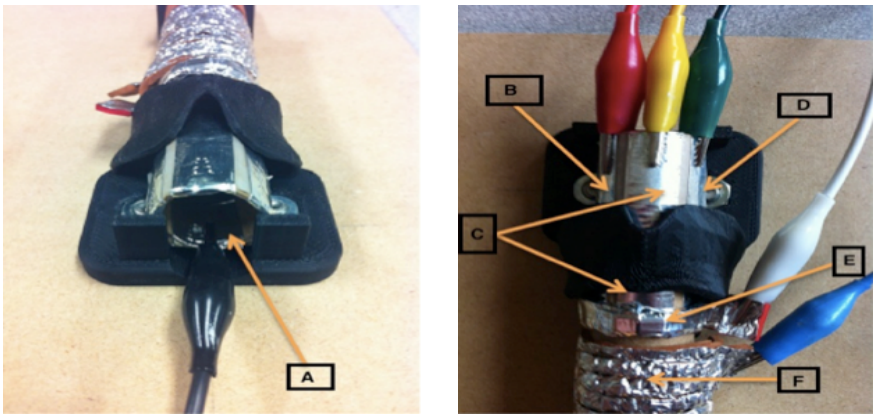



**Figure 77: Critical locations on the trachea are covered with conductive foils that act as cheap sensors to: (A) posterior tracheal wall, (B) right lateral trachea and cricothyroid membrane, (C) midline cricothyroid membrane (the correct points of incision for cricothyrotomy), (D) left lateral trachea and cricothyroid membrane, (E) cricoid cartilage, and (F) cartilaginous ring of lower tracheal wall. Photo credit: UW BioRobotics Lab, Seattle, USA.**

Conductive foils are used as low-cost sensors to detect instrument contact with six critical locations on the trachea that users might contact during the procedure. These locations, covered with conductive foils, are shown in Figure 77. The cricothyroid membrane (Figure 77 (C)) is the only correct location for making the incision. Other locations like – the posterior tracheal wall (A) and lateral locations into the tracheoesphageal grooves (D) – should be avoided. A bicycle inner tube is draped over the trachea's model to simulate human skin (this is shown in Figure 79 below). The instruments used to perform cricothyrotomy are show in Figure 78; these include a scalpel, tracheal hook, and hemostat.

**Figure 78: Instruments used to perform cricothyrotomy. Left to right: scalpel, tracheal hook, and hemostat.**

The six conductive foils and the three instruments are connected to different pins of FoneAstra's processor (c.f. Section 2.3). The complete setup of the simulator is shown in Figure 79.

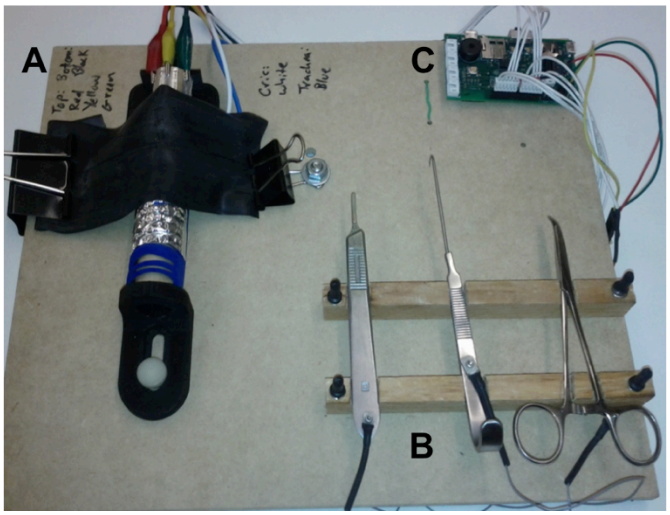



**Figure 79: The cricothyrotomy simulator mounted on a wooden board. (A) The trachea model with 6 conductive foils connected to FoneAstra (C). A bicycle inner tube is draped over the trachea's model to simulate human skin. (B) The instruments used to perform cricothyrotomy, also connected to FoneAstra. (C) The FoneAstra board that records whenever connections are made or broken between an instrument and parts of the trachea. These events are streamed in real-time to an Android device (not shown) over a Bluetooth connection.**

Traditional matrix scanning techniques are used in FoneAstra's software to detect connections between instruments and conductive foils. Every time a closed circuit is detected, the event is recorded and labeled with a timestamp. Similarly, breaking a contact is also recorded and time stamped. Events thus include the instrument, the conductive foil, and whether a connection was made or broken. These events are transmitted in real-time to a co-located Android device over a Bluetooth connection (provided the FoneAstra has already been paired with an Android device and a Bluetooth connection exists between the devices).

On the Android side, an ODK Sensors driver parses sensor data received from FoneAstra via the ODK Sensors framework. It sends this data to a top-level application that provides feedback to guide users through each step of the cricothyrotomy process. In addition to providing real-time feedback to guide users, the application also includes learning materials, such as a video and a slide-deck that describe the cricothyrotomy process.

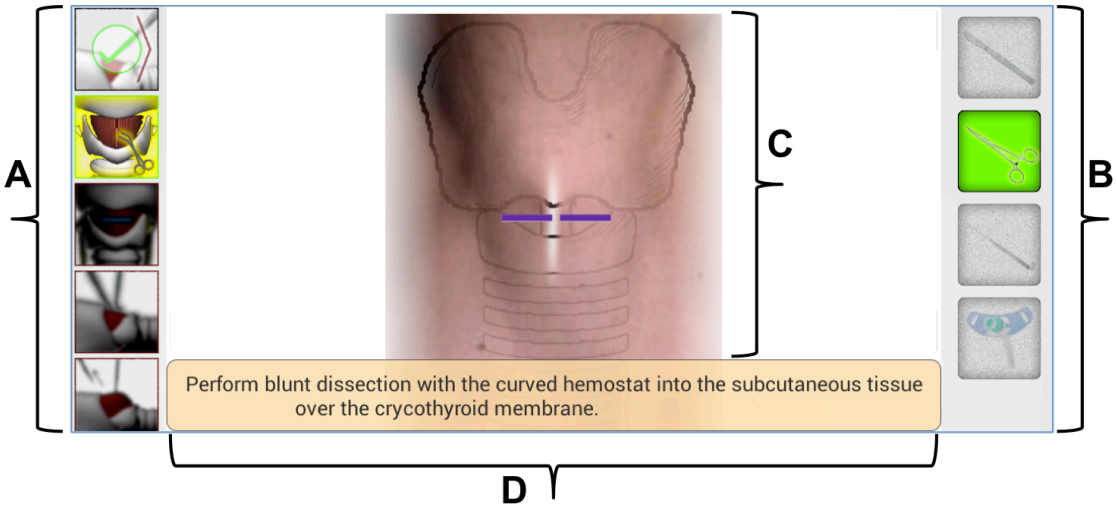



**Figure 80: Screenshot from the Android application that guides users through a cricothyrotomy. (A) Boxes showing the different steps of cricothyrotomy. The current step is highlighted with a green box that acts as a cursor. (B) Boxes showing the different instruments used for cricothyrotomy. The instrument needed for the current step is highlighted in green. (C) Image of the human trachea, the part of the trachea where a step needs to be performed gets highlighted to provide guidance (e.g., the horizontal, purple line on the image). (D) Optional instructions to perform the current step of the procedure.**

Figure 80 shows an annotated screenshot of the application while its being used to perform a cricothyrotomy. The left side of the screen (labeled A) shows the different steps of the cricothyrotomy. The current step to be performed is highlighted with a green box that acts as a cursor (the second box in the column for this particular procedure). If a step is completed

successfully (i.e., if an instrument is used correctly), it is overlaid with a green check mark (the first box of the column in the screenshot), and the cursor moves to the next box. Errors are highlighted with a red cross, and the user is prompted to repeat the step. The right side of the screen (labeled B) shows the different instruments used for the cricothyrotomy. The instrument(s) needed for the current step are highlighted in green (the second box of the column in the screenshot). The middle part of the screen shows the human trachea (labeled C), and the bottom part of the screen (labeled D) provides instructions to the user for performing the current step of the procedure (instructions can be disabled if needed). To provide additional guidance, the part of the trachea where a step needs to be performed is also highlighted (e.g., the horizontal, purple line). If users need more help with a particular step at any point in the procedure, they can view a video snippet of instruction appropriate for that step. The mobile application includes a help screen that explains the different parts of the screen shown in Figure 80 and provides navigation links to learning materials. We evaluated this system with three physicians (or expert users) and eight Computer Science and Engineering graduate students (or novice users). We wanted to understand the usability of the mobile application and its efficacy as a training tool. Additionally, with expert users we wanted to evaluate the correctness of our training features. All experts were familiar with the simulator because they had evaluated an older version (that did not have FoneAstra or the mobile application) with the EE researchers. In clinical settings, they had performed 100 to 400 cricothyrotomy procedures on patients. The novices were not familiar with the simulator and did not know how cricothyrotomies were done.

Before using the system, each participant was shown the cricothyrotomy video and informational slides. They then spent some time familairizing themselves with the mobile application's user interface and understanding its features by using the help screen. We explained the simulator to novices since it was new to them. Participants were allowed to do some practice runs with the simulator and the mobile application until they got comfortable with the system. After this, they performed their final run in which all sensor data received from the simulator was logged to a file on the Android. Finally, they were asked to provide feedback via a survey. The survey had questions about the usability of the mobile application that could be answered using a 6-point Likert scale (1 being the lowest score and 6 being the highest) and a few subjective questions. The survey for experts had an additional section to evaluate the correctness of the training features of the mobile application. These surveys are available in Appendix C.

| Accuracy (rated only by experts) |
|---|
| The mobile application demonstrates the correct way to perform the medical procedure |
| **Usability (rated by experts and novices)** |
| The simulator is useful to practice the procedure |
| The simulator becomes easier to use with practice |

**Table 16: Survey items that are discussed in this section. Only experts rated the accuracy of the system, while both experts and novices rated its usability. The complete surveys are available in Appendix C.**

We now summarize reponses to some survey items (i.e., those listed in Table 16). The experts agreed (average score of their response was 5) that the mobile application demonstrated the correct way to perform cricothyrotomy and that the application along with the simulator, was a useful tool for practicing the procedure. Likewise, the novices agreed (with an average score of 5.4) that the whole system was useful for practicing the procedure. They indicated that it became easier for them to use the system over time (average score of 5.1). Two experts and several novices found it distracting to look at the mobile application to determine if they performed a step correctly on the simulator. We addressed this issue, and the application now provides audio feedback in addition to visual feedback.

Overall, we received a positive response from the participants. We planned to corroborate their self-reported data with the sensor data recorded in each participant's final run; however, this remains to be done. Our colleagues in the EE department continue to use the mobile application and the simulator for their research, collecting data from users who have a medical background. They envision that this sytem could help evaluate the performance of users in addition to its teaching function.

## 7.2 Cardiac Monitoring

A Lancet Commissions' survey on technologies for global health [103] identifies cardiovascular ailments to be amongst the leading causes of disability-adjusted-life-years in developing countries. A report from Partners in Health [104] discusses chronic care integration for Non-Communicable Diseases in the Rwandan rural healthcare system; the challenges, techniques and protocols described there are generalizable to many low-resource settings. This report identifies high equipment costs and a scarcity of adequately trained staff as major challenges that hinder diagnosis and treatment monitoring at lower levels of the healthcare

system. Lack of resources at lower levels can also affect healthcare delivery at higher levels if patient screening, referrals and follow-up treatment monitoring does not happen in a timely manner. Guided by these studies, we are developing mobile, sensor-based tools to improve screening, referrals, monitoring and management of cardiovascular ailments.

Monitoring blood pressure (BP), oxygen saturation (O2) levels in blood, and heart rate provide basic indicators of cardiovascular health. Automatic, low-cost instruments to measure BP, blood O2 levels, and heart rate already exist; they minimize the training needed by health workers at lower levels of the healthcare system to use these instruments correctly. However, diagnosing major ailments (e.g., cardiomyopathy, arrhythmia, etc.) requires sophisticated equipment to obtain patients' EKGs and echocardiograms. Such equipment are expensive, and also require significantly more training to administer tests and interpret results. Hence, it has not been practical to have EKG or echocardiogram equipment at lower levels of the healthcare system.

**Figure 81: An early prototype of FoneAstra's single-lead EKG daughterboard. EKG data is sent from FoneAstra to a co-located Android over Bluetooth when a subject places their thumbs (or fingers) on the two steel plates. Software (currently under development) on the Android processes the sensor data to detect anomalies in the EKG.**

We are developing a low-cost daughterboard for FoneAstra that will enable frontline health workers to easily obtain a patient's EKG. Rather than a 10-12 lead EKG system that requires leads to be placed on different parts of the body, our single-lead EKG system will provide measurements if patients simply hold the device in their hands. This will significantly simplify the measurement of EKG. Our accessory is based on the AD8232 [105] signal-conditioning block for EKG applications. Figure 81 shows an early prototype of the accessory that will plug in to a FoneAstra. The EKG readings are streamed over Bluetooth from FoneAstra to a co-located Android device that receives the data via ODK Sensors.


**Figure 82: Android screenshot of the EKG trace from a subject.**

We are developing algorithms on Android to detect anomalies in EKG readings (e.g., arrhythmias). When integrated with Clinical Decision Support Services (that also use BP and pulse oximetry data as inputs), this work will become part of a patient-screening and referral management tool for frontline health workers. Additionally, the system will enable health workers and remotely located specialists to monitor patients undergoing treatment (e.g., recovering from surgery or currently under medication). Figure 82 shows an Android screenshot of the output from our EKG accessory.

After talking to physicians we have learned that blood O2 levels are highly correlated to a patient's cardiovascular health, and changing O2 levels are also indicative of how a patient is responding to treatment. Therefore, in the near future we will add a low-cost pulse oximeter to our EKG accessory in order to provide a more integrated solution.

## 7.3 Prosthetic Monitoring for Gait Analysis

People with lower-limb amputations often struggle to regain a natural and efficient gait pattern with their prostheses. Analyzing the gait of amputees can help guide rehabilitation therapy and prosthetic fitting. However, classical gait analysis commonly requires a well-equipped laboratory, which prohibits the long-term monitoring of subjects' gait in a natural environment. Rehabilitation and the prosthetic fitting process can be improved by using portable

gait monitoring devices capable of measuring gait patterns over long periods of time. Motivated by this need, researchers in the department of Bioengineering (BioE) at the University of Washington wanted to build a wearable, mobile phone-based sensing system to do continuous gait analysis of people with lower-limb amputations. We collaborated with them to develop a mobile sensing system that leveraged a USB Bridge (cf. Section 2.2.2) and ODK Sensors.

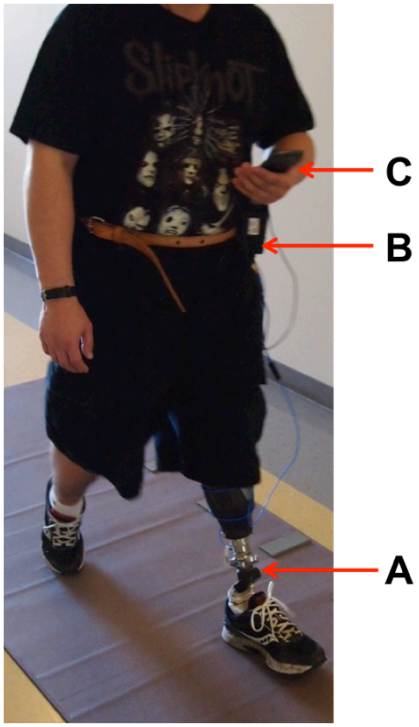



**Figure 83: A subject wearing a prosthesis that has been modified to include the force sensor (A). The sensor is connected to the USB Bridge (B) via a blue wire. The USB Bridge, worn as a belt-pack, is connected to the Android device (C) in the subject's hand. The subject's face is hidden due to privacy concerns. Photo credit: Sanders' UW Research Lab, Seattle, USA.**

The BioE researchers wanted to experiment with a 3-axis kinetic force sensor and an accelerometer for gait analysis. They had already identified the force sensor they wanted to use and we jointly decided to use an off-the-shelf, 3-axis accelerometer for this work. The force sensor is installed between the prosthesis' socket and foot, and the accelerometer is attached at the same location. These sensors are connected to the USB Bridge, which in-turn is connected to an Android device over USB. Figure 83 shows a subject whose prosthesis was modified to include the force sensor; the accelerometer was not used for this subject-test. The USB Bridge is attached to a belt-pack worn by the subject, who is carrying the connected Android device in his hand. Software on the USB Bridge samples the sensors and sends aggregated readings to the

Android device over the USB connection. We implemented two separate ODK Sensor drivers to parse data from the force sensor and the accelerometer.

Figure 84 plots the data acquired from the force sensor while the subject was walking. The subject's prosthetic leg-swing (when the foot is not on the ground) and stance (when the foot is on the ground) phases were determined by thresholding the axial force signal. If the force exceeded the threshold, then the subject's prosthetic leg was in stance phase. If not, the leg was in swing phase. The threshold used to discriminate stance and swing was determined by manually inspecting the force data.

[106] provides more details about the prosthetic gait analysis research done by the BioE team. They compared gait data received from the force sensor to data collected by a GAITRite [107] electronic walkway and saw a strong correlation in the stance and swing phase timing measures. These results supported the intended use of the kinetic sensor for real-time gait monitoring in clinical rehabilitation.

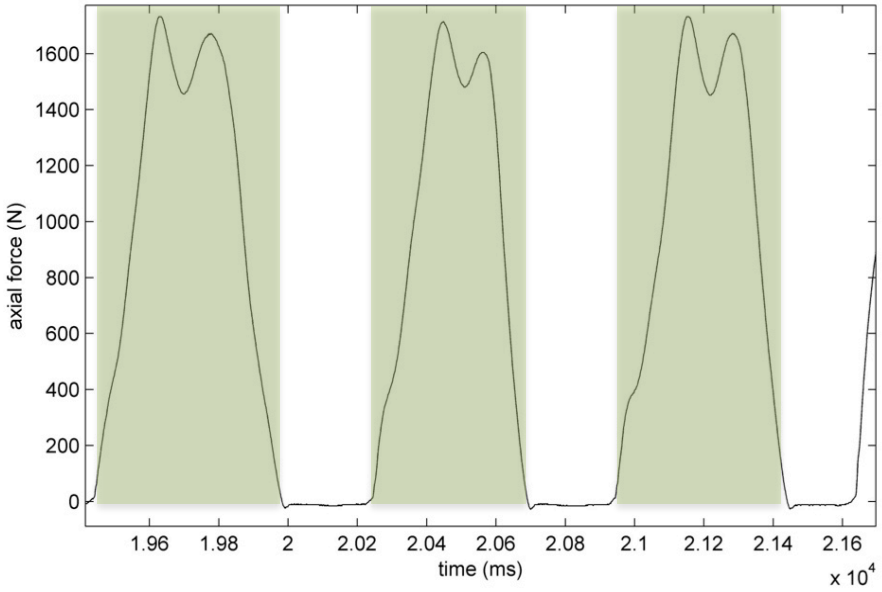



**Figure** 84**: The force profile when the subject was walking, constructed from the data received by the force sensor attached to the prosthesis. Areas highlighted in green are the stance phase of each step; this is the phase of a step is when the foot is on the ground. The initial contact of the foot on the ground causes a sharp rise in force. There is a short dip during this phase when the foot is rolling over from heel to toe. Finally, there is a sharp drop in force when the foot leaves the ground. Image credit: Sanders' UW Research Lab, Seattle, USA.**

## 7.4 mPneumonia

The Integrated Management of Neonatal and Childhood Illnesses (IMNCI) [108] is a paper-based medical protocol specified by UNICEF to help diagnose childhood pneumonia. In the mPneumonia project, we developed a mobile device-based system to guide health workers through the protocol. In this system a pulse oximeter is connected directly to the Android device over USB, and ODK Sensors facilitates communication with the sensor. Sensor data is integrated with data collection forms that are part of an ODK Survey [8] that guides the health worker. Figure 85 shows a picture of our system being used in a field trial. The Android acts as the USB Host in the mPnuemonia system and powers the pulse oximeter to access it as a USB-serial device. Integrating the USB pulse oximeter with Android using ODK Sensors also led to the creation of a new USB sub-channel within the ODK Sensors framework, letting it interface with USB-serial devices (cf. Section 3.3).

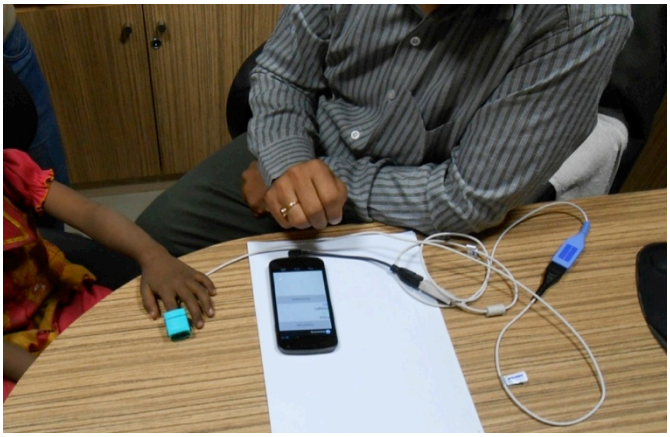


**Figure 85: A clinician using the mPneumonia Android app to get pulse oximetry data from a child wearing the USB pulse oximeter on a finger. Faces are hidden due to privacy concerns. Photo credit: Waylon Brunette, UW, Seattle, USA.**

## 7.5 Conclusion

This chapter described several applications that use the technology building blocks presented in Chapters 2 and 3. These applications provide evidence that the underlying technologies developed in this thesis can be used to develop a wide variety of mobile sensing applications. The emergency tracheotomy training system was developed in collaboration with a research group in the department of Electrical Engineering at the University of Washington. We got positive results from evaluations of this system done by both physicians and graduate students who didn't have any medical background. Our collaborators continue to use this system for their research. The cardiac monitoring project is a work in progress within our group, and we

intend to start evaluating it soon. In collaboration with researchers in the department of Bioengineering at the University of Washington, we developed a wearable system to continuously monitor the gait of people who wear a prosthetic leg. Our collaborators evaluated this system with one subject to date. Finally, the mPneumonia project is meant to assist health workers diagnose pneumonia in children.

# Chapter 8

# Conclusion

Mobile devices like smartphones and tablets have evolved from being mere communication devices to being an attractive computing platform with myriad applications. In conjunction with cloud-hosted services, they are increasingly being used to deliver information services in low-resource settings, such as in developing countries. They already have a variety of built-in sensors (e.g., accelerometer, gyroscope, GPS, camera etc.), enabling a wide range of context-aware mobile applications. Additionally, the built-in communication interfaces like WiFi, Bluetooth and USB let mobile devices communicate with external devices and sensors.

Sensors connected to mobile devices help bridge the physical and digital worlds by providing precise readings on various phenomena (e.g., environmental and soil conditions). However, creating mobile sensing applications poses several barriers for developers, namely:

- Sensors often have only low-level communication interfaces; hence, an interfacing board is needed to physically and logically connect them to mobile devices.
- Partitioning a sensing application between a mobile device running an OS (like Android or iOS) and an interfacing board that likely has an embedded OS at best, adds complexity to the system and requires a less common skillset.
- In addition to implementing application logic, developers must address quirks of different physical communication channels and process sensor-specific data.

My thesis addresses these problems by extending the sensing capabilities and modalities of mobile devices to simplify the development of mobile sensing applications. The following statement summarizes my thesis: ***Simplifying connections between mobile devices and sensors leads to better measurements that can improve global health systems.***

This thesis makes the following three contributions:

1. Hardware to enhance and simplify a mobile device's ability to connect to various types of sensors ([4]–[7])
2. A software framework to simplify development of mobile sensing applications ([5], [6], [8])
3. Implementation and evaluation of sensing systems to enhance workflows in low-resource settings ([9]–[13])

Chapter 2 covers our work on different approaches to connect sensors with low-level interfaces to mobile devices; it provides evidence for the first contribution of this thesis. We developed a few interfacing boards that act as a bridge between such sensors and mobile devices. Early work focused on extending the capabilities of low-tier, non-programmable mobile phones via an interfacing board called FoneAstra. FoneAstra hosts the application-specific sensors and code and connects to a phone over a serial interface. It leverages the phone to communicate with backend servers or other mobiles and provides coarse-grained location tracking based on the cell tower IDs visible to the phone. This platform was used to monitor some parts of the vaccine cold chain in Albania and Nicaragua. Motivated by application needs, we developed the second version of FoneAstra, which enhanced its feedback capabilities. The new board included a 2-line LCD, an audio buzzer, more prominent LEDs, and other enhancements. This was the first platform used in the human milk-banking project.

As Android devices became cheaper and more practical for use in developing countries, we switched to using Android for our mobile sensing work. However, an interfacing board for low-level sensors was still needed. We developed two interfacing boards to extend the sensing capabilities of smartphones. One of the boards communicates with Android over a USB connection; it was part of the second platform used in the human milk-banking project. The other board communicates with Android over a Bluetooth connection; it was used in Ethiopia to measure the time spent by rural residents in collecting water for daily use.

Insights gained from developing these interfacing boards led to the creation of the third and final version of FoneAstra. This board combines the capabilities of the previous boards into one package and supports different kinds of mobile sensing applications. It communicates with a mobile device over Bluetooth or USB and can be powered by a dedicated battery or via USB by devices that have USB Host capability. It provides dedicated ports for connecting 1-wire, $I^2C$,

SPI, and UART devices, and all the processor pins are accessible on connection headers for interfacing with devices that might require some other interfacing method (e.g., an analog interface).

FoneAstra V3 is suitable for mobile sensing applications that process sensor data in real-time (e.g., vaccine cold chain monitoring and human milk banking), or applications that periodically retrieve historical sensor data (e.g., time-use measurements). FoneAstra is also being used in several other applications (discussed in Chapter 7) and by other groups in the community for their mobile sensing work. To simplify the deployment of mobile sensing applications, we developed an Android application that reprograms FoneAstra over a Bluetooth connection. This lets application developers include the board's firmware in an Android application that is published in app markets such as Google Play. We believe that this will simplify the distribution and maintenance of firmware for boards like FoneAstra and will be beneficial for sensor manufacturers, application developers and end-users alike.

Chapter 3 describes the Open Data Kit (ODK) Sensors framework, an application-level sensing framework for Android. This work provides evidence for the second contribution of the thesis. Currently, mobile sensing applications are developed as monolithic systems that include the application logic and functionality to communicate with sensors and process sensor-specific data. ODK Sensors simplifies the development of such applications by providing abstractions that segregate application logic, communication channel-specific functionality, and sensor-specific data processing. These abstractions simplify the creation and maintenance of sensing applications and enable a high level of customization and flexibility, thereby permitting a variety of wired and wireless sensors to be connected to mobile devices. Sensor-specific data processing is done in sensor drivers, which are implemented as user-level Android apps that can be developed and distributed independently.

The ODK Sensors framework is implemented as another user-level app that encapsulates communication channels within different channel managers; these managers implement channel-specific functionality to allow communication with devices over their respective channels. As of 2014, the framework has channel managers for Bluetooth and USB. It further simplifies the development of sensor drivers by managing sensor and connection states, data buffers, and threads, thereby reducing the driver code needed to handle sensor-specific commands and data processing. Top-level apps implement domain-specific use cases and interface with the

framework via a unified API that enables these apps to discover, connect, start and stop sensors, and receive parsed sensor data. Since all apps are implemented at the user-level, they can be installed and used on consumer Android devices that may be locked.

To create a reusable ecosystem of sensor drivers, the framework leverages standard app marketplaces (such as Google Play); this lets end-users find/install drivers and perform automatic updates as they do for any other Android app. End-users can get a complete sensing system on their devices by simply aggregating the required components that may be written by different developers. After obtaining the sensing hardware, a user needs to download and install three android apps on their device: (1) the ODK Sensors framework, (2) a sensor driver that could include firmware for interfacing devices like FoneAstra, and (3) the top-level sensing app that implements the use case. The top-level app presents the framework's user interface to guide the user through the sensor discovery and driver selection process before it can receive parsed sensor data from the framework.

Chapters 4, 5, and 6 describe mobile sensing applications developed using these technology building blocks and that helped guide their evolution. These applications provide evidence for the third contribution of the thesis. Chapter 4 covers our work on a real-time vaccine cold chain monitoring system that uses FoneAstra and low-tier mobile phones. We achieved promising results from a six-month trial of this system in Albania. The Albanian cold chain staff identified poorly performing equipment based on the temperature profiles of equipment accessible from the system's server. Lessons learned from the trial in Albania led to the development of a more scalable cold chain monitoring system that leverages ODK Sensors and FoneAstra (V3); this is a work-in-progress as of this writing.

Chapter 5 presents our work on managing the pasteurization of breast milk at human milk banks. We developed a mobile sensing system using ODK Sensors and FoneAstra (V3) for this use case. Our system monitors the temperature of milk while it is being pasteurized and provides feedback to users to guide them through the process. After the pasteurization is complete, the system lets the user print labels for milk bottles via a Bluetooth-enabled printer. We started system deployments at human milk banks in South Africa in 2012. As of 2014, it is installed at five locations in the country, and we are working to scale it up further in collaboration with our project partners.

Chapter 6 describes a mobile sensing system that allows researchers working in the water sector to get accurate data about the time rural residents spend collecting water for daily use (called time-use). We built a Bluetooth-enabled motion sensing board that, when attached to a water container, records the times when the container is in motion. Researchers periodically retrieve the recorded data via a Bluetooth-enabled Android device. This data is then aggregated on a computer for analysis. This system was trialed in three villages in Ethiopia in 2011. It represents a use case where a sensor is deployed autonomously, and encountered only occasionally to transfer its data.

Chapter 7 presents other applications that leverage the technologies described in Chapters 2 and 3. These applications demonstrate that the underlying technologies are generalizable for use in a wide variety of mobile sensing applications. One application is a wearable, real-time prosthetics monitoring system that was trialed by our collaborators in the UW Department of Bioengineering. Another application is a low-cost, mobile device-based simulator for the emergency tracheotomy procedure, now being used by our collaborators in the UW Department of Electrical Engineering for their research. We also developed an application that uses pulse oximetry to help with the diagnosis of childhood pneumonia. Finally, we are developing: (1) a low-cost sensor board that will plug into FoneAstra to measure a subject's EKG and blood oxygen saturation level, and (2) an Android application that will interpret data from the sensor board and act as a screening tool for frontline health workers to help them identify at-risk patients in rural areas.

## 8.1 Insights on Creating Technologies for Developing Regions

Researchers in the ICTD community have shared their experiences from deploying technologies in developing regions (e.g., [109], [110]). In this section, I discuss some lessons I learned while working in the ICTD field. I draw specific examples from our human milk banking work to provide context.

### 8.1.1 Needs Assessment and Workflow Integration

Our work tends to be multidisciplinary, and we often work with geographically dispersed organizations. If technology development occurs in a developed country (which has been the case for the applications developed in this work), we typically connect with our target users/organizations in developing countries through a partner organization (e.g., PATH). Our

partners and end-users provide the requirements that need to be addressed for a problem and help guide the development of the solution. Therefore, frequent and effective communication with all stakeholders is very important. For instance, in the milk banking project, we started sharing artifacts (such as screenshots of the mobile application and the supporting website) long before we traveled to South Africa for the first time. This helped validate that we were on the right track and also helped our partners understand how the system would work.

The initial days of a field trial are very educational. We see operations in the field for the first time and learn more specifics about workflows. This helps us identify more use cases where technology could be introduced to make the current solution integrate better with existing workflows. It is also an opportunity to take a step back and evaluate the appropriateness of our solution because we might also learn that certain features are not required or must be adapted/refined further. It is helpful to think about questions such as: How does our technology affect the workflow? Does it make things easier or more difficult for system users? Does it introduce any new processes/activities in the workflow? Such questions can help understand the barriers to adoption of the technology, an important factor to consider for the long-term sustainability of the intervention. In the milk banking project, we added printing capability to our system based on field observations; this meant that users had to enter additional information in the mobile app (an increase in difficulty), but printing obviated the need for handwritten records and bottle labels (a more significant simplification of the process for users).

### 8.1.2 Cost

The cost of a solution is an important consideration, but focusing primarily on minimizing the cost might not always be optimal. Solutions should be built upon a flexible, reusable set of core technologies that make it easier to address new requirements as we learn more about the problem-domain (e.g., during field trials). The evolution of our technology for human milk banks provides a good example. The first version of our pasteurization monitoring system leveraged low-tier phones and FoneAstra V1; it was a cost-effective solution to guide users through the pasteurization process. However, having a low-tier phone proved to be limiting, and it would have been difficult to extend the system to do anything more than just monitor pasteurization temperatures and provide audiovisual feedback. The second version of the system leveraged ODK Sensors and Android, which raised system cost but provided flexibility, which made it possible to rapidly modify the system to address new requirements we identified

in the field (e.g., adding printing capability to the system). Addressing these requirements was critical because it integrated the technology better with the workflows at human milk banks.

### 8.1.3 Technology in the Bigger Picture

The technology solutions we develop typically address specific problems that are part of a larger societal problem. For instance, human milk banks can help lower infant mortality, and our technology for human milk banks manages breast milk pasteurization that is a critical part of the larger human milk banking process (cf. Section 5.2). Scaling up an intervention and making it truly sustainable often requires working with policymakers to affect country-level policies in order to achieve the desired impact. This is true for our human milk banking work in South Africa. Therefore, it is important to have partners like PATH who work with governments to affect policy-level changes.

### 8.1.4 Business Strategies

Compared to mainstream markets (e.g., mobiles, wearables, etc.) that are huge and grow rapidly, markets for most of our developing country technologies are small and do not grow as rapidly. This makes it difficult for technology providers to have a sustainable business model. Targeting multiple markets, including developed-country markets, could be one way to work around this problem. Here again, it helps to build upon a common, reusable set of core technologies to pursue different markets/applications rather than to build solutions from scratch for each market. The increasing market penetration of mobile devices [41] makes them attractive for use as a base platform upon which common, reusable core technologies can be built. Such devices can now be used to develop applications that not only interact with humans, but that also make the device act as an "appliance" (e.g., the applications discussed in Chapters 4 and 5), which further increases the markets for mobile technologies.

### 8.1.5 Other Considerations

There are some common patterns in our deployment contexts. For instance, appropriate diagnostics equipment and/or trained personnel might not be available at certain levels of the healthcare system, our target communities might be semi- or illiterate, infrastructure (e.g., electricity, Internet connectivity) might be unreliable or unavailable, etc. We could leverage work from other areas of Computer Science to address some of these problems. For instance, computer vision-based systems implemented on mobile devices could be used to interpret

diagnostic test results to identify diseases in humans [111]. Such systems could also be used in the agriculture sector to identify crop diseases. Novel sensing systems developed by the ubiquitous computing community could help improve healthcare delivery (e.g., SpiroSmart [112]). Gesture-based interfaces developed by the same community could help address literacy-related barriers to technology adoption. Networking researchers have invented ways to harvest power (e.g., [113]), enabling devices to operate and communicate without an onboard power-source. These could prove to be useful in our deployment contexts.

## 8.2 Future Work

This section presents a few promising directions of work that can build upon my thesis.

### 8.2.1 Firmware Generation for FoneAstra

The FoneAstra programmer (discussed in Section 2.3.1) is an Android application that can install new firmware on the FoneAstra board over a Bluetooth connection. This simplifies the deployment and maintenance of mobile sensing applications that leverage interfacing boards such as FoneAstra. The development of such applications could also be simplified by creating a tool to generate firmware based on a high-level specification of the board's runtime behavior. For instance, a high-level language could be defined that allows a user to specify the sensors needed for an application along with certain parameters (e.g., sensor connection interface/pins, sampling rate, buffer size, how/when to transmit sensor data to a mobile or store the data locally, etc.) in a spreadsheet. The tool could take this spreadsheet as input, convert it to C code, then compile and link the C code to generate a firmware file. Such a tool could enable non-programmers to create applications for their own needs.

### 8.2.2 Mobile Device-based Medical Procedure Simulators

The Emergency Tracheotomy training tool (Section 7.1) could be extended to create a system to teach the non-emergency tracheotomy procedure that requires intubation. This general approach can also be applied to create a set of training systems for other palpation-based medical procedures, such as inserting intravenous needles or administering CPR.

### 8.2.3 Diagnostics

The increasing prevalence of diabetes in developing countries [114] has created the need for rapid, low-cost and noninvasive diagnostics for diabetes risk assessment. Advanced Glycation Endproduct (AGE) readers [115] are a promising technology to accomplish this. Such

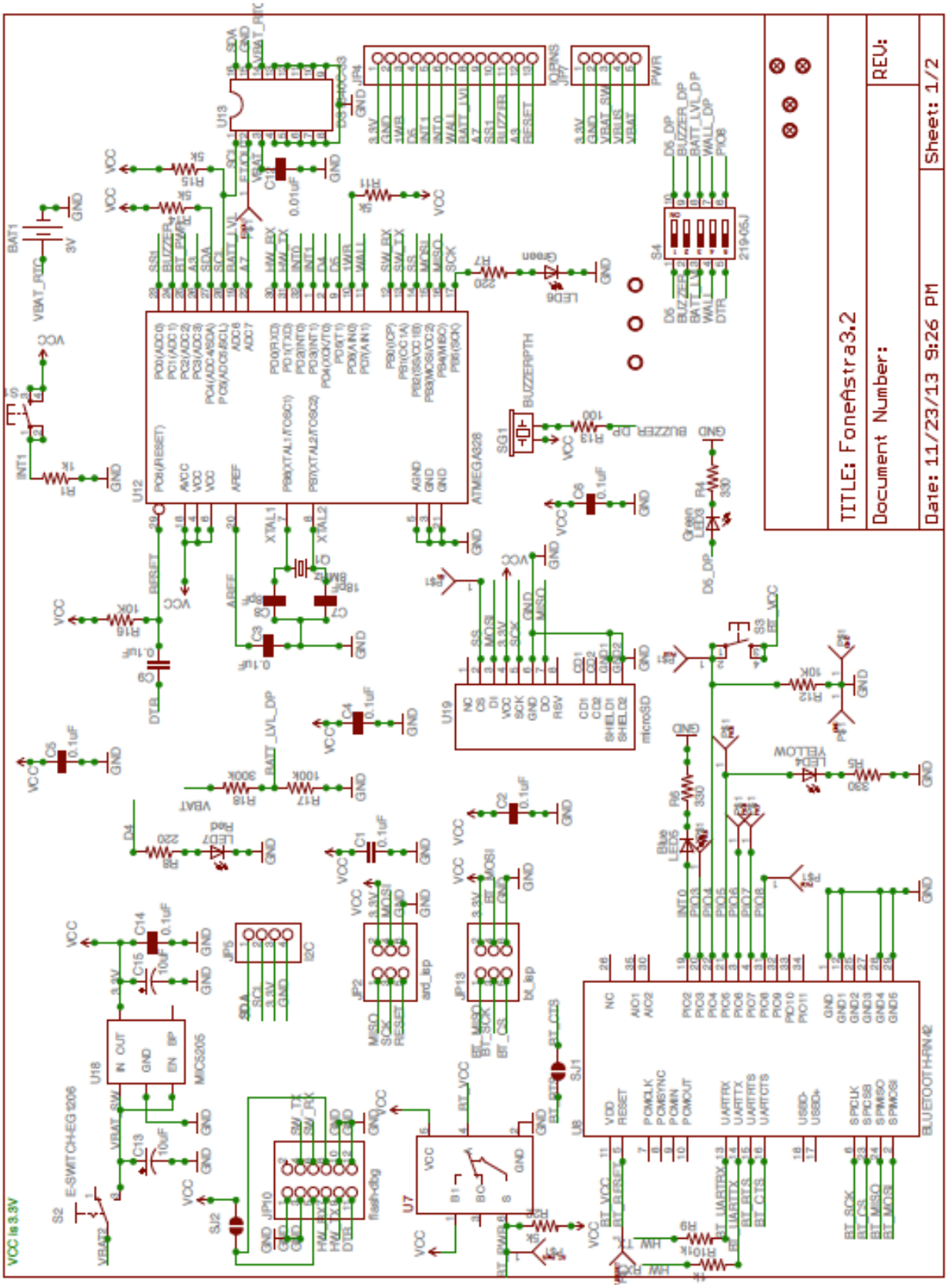
devices measure skin fluorescence due to the accumulation of diabetes related AGEs in the skin, as well as other biomarkers of metabolism and oxidative stress, to determine a subject's risk for diabetes, cardiovascular or renal ailments. An AGE reader has an ultraviolet light-source that illuminates some part of a subject's skin (e.g., the forearm). This light excites fluorescent molecules in the skin tissue, which then emit light of a different wavelength. The wavelength and intensity of the emitted light are indicators of the risk of having diabetic, cardiovascular or renal ailments. However, such readers are expensive, which makes their adoption difficult at some levels of the healthcare system in developing countries.

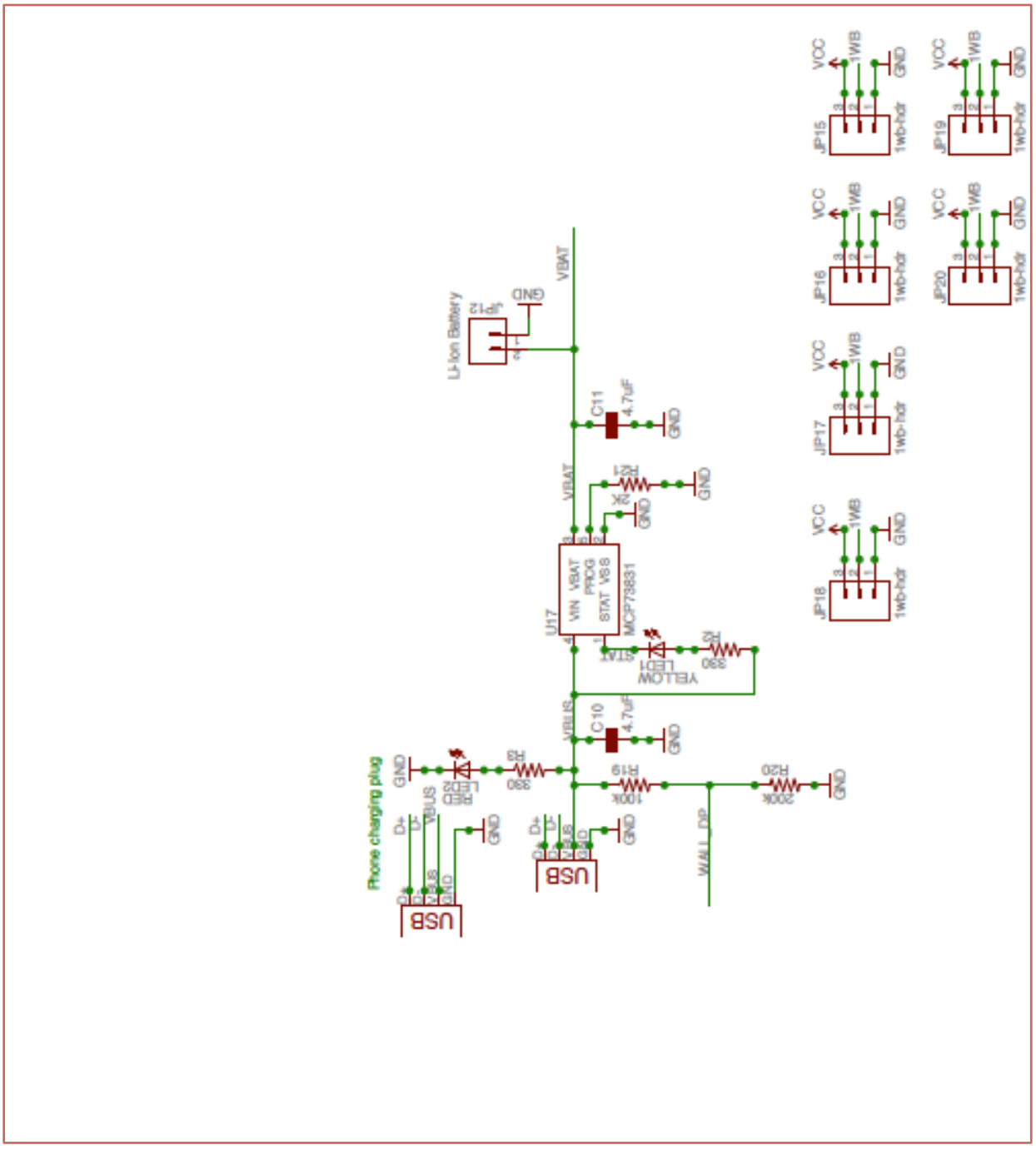
It might be possible to create a more affordable AGE reader that leverages a low-cost ultraviolet light-source connected to a mobile device. Software on the mobile device could analyze skin fluorescence to determine risk factors. Additionally, diagnosis results could be integrated with patient records and clinical decision support services, letting less skilled health workers effectively use the mobile AGE reader.

## 8.3 Final Remarks

My thesis has developed hardware and software mechanisms to expand the sensing capabilities of mobile devices. Several applications developed using these mechanisms validated this work. The hardware and software artifacts have been open-sourced, and it is my hope that the community will leverage and expand this work further.

# Appendix A: FoneAstra 3.0 Schematic




VCC
1WB
GND
JP15
JP16
JP17
JP18
JP19
JP20
1wb-hdr
VBAT
Li-Ion Battery
JP12
C11
4.7uF
R1
1K
U17
VIN
VBAT
PROG
STAT
VSS
MCP73831
STAT
LED1
YELLOW
R2
330
VBUS
C10
4.7uF
R3
330
LED2
RED
Phone charging plug
R19
100K
R20
200K
USB
D+
D-
VBUS
GND
WALL_DC

# Appendix B: Baseline Interview Questionnaire for Milk Bank Staff

## *Demographics*

**Name**:

**Age** (don't need exact age, ask to select appropriate range from choices of 10-year ranges):

**Gender**:

**Title** (if any):

**Department**:

**Educational background:**

**What do you do (what tasks do you perform, how are you affiliated with milk banking)?**

## *Role in milk banking*

**Describe your role with the Human Milk Bank (HMB).**

**Have you used any other milk pasteurization systems? If yes, what systems? Please describe.**

**HMBASA has several milk banks, what other milk pasteurization systems do they use? (For HMBASA supervisors)**

## *Data collection for baseline*

**Why do you perform Flash Heat Pasteurization (FHP)?**
**(Follow-up: Is it OK if milk is under- or over-heated?)**

**What training did you receive for learning FHP?**

**What equipment do you use for performing FHP?**
(e.g., jar, pan, stove, measuring device)

**How do you perform FHP?**
(i.e., you already have donor milk, maybe in a jar, what next?)**.**

**What is most difficult about the FHP process? Could any mistakes be made during the process?**

**When do you know that milk should be removed from the water-bath?**

**How do you know water is at a rolling boil?**

**How do you feel about the boiling water as the indicator to remove milk from the water-bath (like a visual cue)?**

**Do you think there is a difference in the milk after heating? Why? Does it look or smell different?**

**After heating, how or what makes you feel confident that the milk is safe/good enough to be fed to babies?**

**Are you able to tell whether the heating procedure was done correctly? How?**

**If something went wrong with the heating, what do you do?**
(Want to understand if they have any notion of "quality" of the procedure. Is this something that supervisors, technicians care for? Is this a concern for recipient baby's family or MOH or orgs like WHO etc.?)

**How do you know that milk is good enough to be fed to a baby?**
(i.e., what are the quality control mechanisms for ensuring high quality breast milk?)

**How do you cool the milk?**

**How do you tell that the milk has cooled down sufficiently?**

**What do you do with the milk after it has been cooled?**
(If it is stored, how is it stored?)

**How long does the whole procedure take?**

**Do you think it takes too long or short to perform the procedure? Or is the time taken just about right?**

**What are the other tasks you do during your shift?**

**How do you schedule FHP? How many per day/week?**

**Where are you when FHP is taking place?**

**Do you ever get interrupted while doing FHP? Is it normal to get interrupted? Give an example of this happening. How did this affect the FHP process?**

**What volume of milk do you process per FHP?**

**Is there a need for increasing the volume of milk processed per procedure?**

**How could the volumes be increased?**
(if subject answers yes to the previous question)

**Is there anything in your current procedure that you would like to change?**

## *Mobile phone usage/familiarity*

**What type of mobile phones are you familiar with?**

**Have you seen phones being used for something other than its telephone (phone calls/SMS) capability?**
(e.g.,: playing games, Internet, email etc.)

**Are you familiar with smart phones?**

**Do you own or use a mobile phone?**

**How long have you used mobile phones (the current one & past mobiles)?**
**(**If the subject answers yes to the previous question)

**Where do you buy your phone?**
(if they own and have purchased phone(s).)

**What kind of mobile phone do you have?**
(ask about manufacturer, model etc. tells you if they care enough to know about such details).

**Who is the cellular service provider?**
(Do they care enough about this?)

**What kind of cellular plan do you have? Do you have a post paid or pre-paid cellular plan?**
(Do they care enough to know about such details?)

**How many mobiles do you have in the household?**

**Does anyone in your family use other kinds of phones? What phones?**

**Do you share your mobile with family members?**

**What do you typically do on your phone?**

How often?

How long?

**Do you access the internet/email on your phone?**

(if appropriate, check to see if they use Mxit, Facebook etc.)

**If the mobile is shared with family members, what do they do with the phone?**

**Would you prefer to use a different type of phone? Why?**

# Appendix C: Post-trial Survey for Tracheotomy Project Evaluation

## Survey for Experts

**Participant # _____**

**Date: _____**

### General Information

| | On a simulator | Clinical (non-emergent) | Emergent |
|---|---|---|---|
| Approximately how many tracheotomies have you performed to date? | | | |
| When was the last time you performed the procedure? (days, weeks, months, years, never) | | | |

### Accuracy of guidance provided by mobile application

### Instructions: Circle the response you agree with most.

**1. The simulator is realistic.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**2. The model is anatomically accurate.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**3. The android tutorial application demonstrates the correct way to perform the surgical operation.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**4. Visuals in the android tutorial application are accurate.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**5. Textual instructions in the android tutorial application are accurate.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**6. Textual warnings in the android tutorial application are accurate.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**7. The procedural steps in the android tutorial application are NOT in the correct order.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**8. Each step in the android tutorial application provides accurate information.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**9. Textual instructions in the android tutorial application need significant improvements.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**10. Textual warnings in the android tutorial application need significant improvements.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**11. I recommend using this mobile application to train for this surgical procedure.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

## Usability of mobile application

## Instructions: Circle the response you agree with most.

**1. The model improved my understanding of how to perform the procedure.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**2. This simulator is useful to practice the procedure.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**3. The simulator is easier to use on the 2nd (or 3rd, or subsequent) time (s). (Leave blank if it's your first time)**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**4. It was easy to use the mobile application.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**5. The on-screen instructions and prompts are helpful.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**6. Learning to use this mobile application is easy.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**7. The mobile application is awkward.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**8. The on-screen instructions are very helpful.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**9. The mobile application feels restrictive.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**10. The button-labels are clear and understandable.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**11. I feel in command of the mobile application when I'm using it.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**12. I understand on-screen instructions.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**13. There is never enough information on the screen when I need it.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**14. Using this mobile application is frustrating.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**15. I felt tense at times when using the mobile application.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**16. There are too many steps required to get something to work.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**17. New users will find this device easy to use.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**18. It is obvious that my needs have been taken into consideration in this mobile application.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**19. The error messages are understandable.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

**20. The mobile application did what I expected.**

[ Disagree Strongly -- Disagree -- Tend to Disagree -- Tend to Agree -- Agree -- Agree Strongly ]

## Additional Feedback

## Instructions: Provide feedback to the following questions. If you need more space, please feel free to write on the back of the page.

1. What aspects of the mobile application are inaccurate?
2. What changes do you want to see for the next version?
3. What was MOST DIFFICULT to do or understand?
4. What was EASIEST to do or understand?

**Please share any additional comments/suggestions.**

## For Developer Use Only

**Notes taken during the subject test:**

**Recurring patterns in errors made by users that we should be thinking about:**

# Survey for Novices

Other than the section about the accuracy of guidance provided by the mobile app, the survey for novices was the same as that for experts.

# Bibliography


[1] "Open Data Kit." [Online]. http://opendatakit.org/. [Accessed: 13-Jun-2014].

[2] "Glympse - Share your location." [Online]. https://play.google.com/store/apps/details?id=com.glympse.android.glympse&hl=en. [Accessed: 13-Jun-2014].

[3] "Noom CardioTrainer." [Online]. https://play.google.com/store/apps/details?id=com.wsl.CardioTrainer&hl=en. [Accessed: 13-Jun-2014].

[4] R. Chaudhri, G. Borriello, and W. Thies, "FoneAstra: making mobile phones smarter," in *Proceedings of the 4th ACM Workshop on Networked Systems for Developing Regions*, New York, NY, USA, 2010, pp. 3:1–3:5.

[5] R. Chaudhri, W. Brunette, M. Goel, R. Sodt, J. VanOrden, M. Falcone, and G. Borriello, "Open data kit sensors: mobile data collection with wired and wireless sensors," in *Proceedings of the 2nd ACM Symposium on Computing for Development*, New York, NY, USA, 2012, pp. 9:1–9:10.

[6] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. Van Orden, and G. Borriello, "Open data kit sensors: a sensor integration framework for android at the application-level," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, New York, NY, USA, 2012, pp. 351–364.

[7] R. Chaudhri, W. Brunette, B. Hemingway, and G. Borriello, "ODK Sensors: An Application-level Sensor Framework for Android Devices," in *Proceedings of the 3rd ACM Symposium on Computing for Development*, New York, NY, USA, 2013, pp. 30:1–30:2.

[8] W. Brunette, M. Sundt, N. Dell, R. Chaudhri, N. Breit, and G. Borriello, "Open Data Kit 2.0: Expanding and Refining Information Services for Developing Regions," in *Proceedings of*

*the 14th Workshop on Mobile Computing Systems and Applications*, New York, NY, USA, 2013, pp. 10:1–10:6.

[9] R. Chaudhri, D. Vlachos, J. Kaza, J. Palludan, N. Bilbao, T. Martin, G. Borriello, B. Kolko, and K. Israel-Ballard, "A system for safe flash-heat pasteurization of human breast milk," in *Proceedings of the 5th ACM workshop on Networked systems for developing regions*, New York, NY, USA, 2011, pp. 9–14.

[10] R. Chaudhri, E. O'Rourke, S. McGuire, G. Borriello, and R. Anderson, "FoneAstra: enabling remote monitoring of vaccine cold-chains using commodity mobile phones," in *Proceedings of the First ACM Symposium on Computing for Development*, New York, NY, USA, 2010, pp. 14:1–14:9.

[11] R. Chaudhri, G. Borriello, and R. Anderson, "Monitoring Vaccine Cold Chains in Developing Countries," *IEEE Pervasive Comput.*, vol. 11, no. 3, pp. 26–33, March.

[12] R. Chaudhri, R. Sodt, K. Lieberg, J. Chilton, G. Borriello, Y. J. Masuda, and J. Cook, "Sensors and Smartphones: Tracking Water Collection in Rural Ethiopia," *IEEE Pervasive Comput.*, vol. 11, no. 3, pp. 15 –24, Mar. 2012.

[13] R. Chaudhri, D. Vlachos, G. Borriello, K. Israel-Ballard, A. Coutsoudis, P. Reimers, and N. Perin, "Decentralized human milk banking with ODK sensors," in *Proceedings of the 3rd ACM Symposium on Computing for Development*, New York, NY, USA, 2013, pp. 4:1–4:10.

[14] E. Cutrell, "Mobile Phone and SMS Survey for LabourNet."

[15] "LPC2148." [Online]. http://www.nxp.com/products/microcontrollers/arm7/LPC2148FBD64.html. [Accessed: 13-Jun-2014].

[16] R. Chaudhri and K. Toyama, "A Programmable Accessory for Cell Phones - Microsoft Research." [Online]. http://research.microsoft.com/apps/pubs/?id=117754. [Accessed: 13-Jun-2014].

[17] “3GPP specification: 27.007.” [Online]. http://www.3gpp.org/DynaReport/27007.htm. [Accessed: 13-Jun-2014].

[18] “FBUS.” [Online]. http://en.wikipedia.org/wiki/FBus. [Accessed: 13-Jun-2014].

[19] “FreeRTOS - Real Time Operating System for embedded systems.” [Online]. http://www.freertos.org/. [Accessed: 13-Jun-2014].

[20] “PATH: Driving transformative innovation to save lives.” [Online]. http://path.org/. [Accessed: 13-Jun-2014].

[21] C. Campo, C. García-Rubio, and A. Cortés, “Performance Evaluation of J2ME and Symbian Applications in Smart Camera Phones,” in *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*, J. M. Corchado, D. I. Tapia, and J. Bravo, Eds. Springer Berlin Heidelberg, 2009, pp. 48–56.

[22] “IDEOS: Android Marches on East Africa.” [Online]. http://www.technologyreview.com/news/424454/android-marches-on-east-africa/. [Accessed: 13-Jun-2014].

[23] “CM1800-1 General purpose metal tilt switch.” [Online]. http://il.rsdelivers.com/product/assemtech/cm1800-1/general-purpose-metal-tilt-switch-120vac/2357572.aspx. [Accessed: 13-Jun-2014].

[24] “Accessory Development Kit.” [Online]. http://developer.android.com/tools/adk/index.html. [Accessed: 13-Jun-2014].

[25] “Arduino Mega2560.” [Online]. http://arduino.cc/en/Main/arduinoBoardMega2560. [Accessed: 13-Jun-2014].

[26] “ATmega328P.” [Online]. http://www.atmel.com/devices/atmega328p.aspx. [Accessed: 17-Jun-2014].

[27] “RN42 Bluetooth Module.” [Online]. https://www.sparkfun.com/products/10253. [Accessed: 13-Jun-2014].

[28] "STK500." [Online]. www.atmel.com/images/doc1925.pdf.

[29] "AVRDUDE - AVR Downloader/UploaDEr." [Online]. http://www.nongnu.org/avrdude/. [Accessed: 13-Jun-2014].

[30] Y.-S. Kuo, S. Verma, T. Schmid, and P. Dutta, "Hijacking power and bandwidth from the mobile phone's audio interface," in *Proceedings of the First ACM Symposium on Computing for Development*, New York, NY, USA, 2010, pp. 24:1–24:10.

[31] S. Verma, A. Robinson, and P. Dutta, "AudioDAQ: turning the mobile phone's ubiquitous headset port into a universal data acquisition interface," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, New York, NY, USA, 2012, pp. 197–210.

[32] "IOIO-OTG." [Online]. https://www.sparkfun.com/products/11343. [Accessed: 13-Jun-2014].

[33] "IOIO Over Bluetooth." [Online]. https://github.com/ytai/ioio/wiki/IOIO-Over-Bluetooth. [Accessed: 13-Jun-2014].

[34] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, New York, NY, USA, 2008, pp. 337–350.

[35] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, New York, NY, USA, 2008, pp. 323–336.

[36] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. Legrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, P. Klasnja, K. Koscher, J. A. Landay, J. Lester, D. Wyatt, and D. Haehnel, "The Mobile Sensing Platform: An Embedded Activity Recognition System," *IEEE Pervasive Comput.*, vol. 7, no. 2, pp. 32–41, April-June.

[37] S. Consolvo, D. W. McDonald, T. Toscos, M. Y. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, R. Libby, I. Smith, and J. A. Landay, "Activity sensing in the wild: a field trial of ubifit garden," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2008, pp. 1797–1806.

[38] J. Lester, C. Hartung, L. Pina, R. Libby, G. Borriello, and G. Duncan, "Validated caloric expenditure estimation using a single body-worn sensor," in *Proceedings of the 11th international conference on Ubiquitous computing*, New York, NY, USA, 2009, pp. 225–234.

[39] B. Priyantha, D. Lymberopoulos, and J. Liu, "LittleRock: Enabling Energy-Efficient Continuous Sensing on Mobile Phones," *IEEE Pervasive Comput.*, vol. 10, no. 2, pp. 12–15, April-June.

[40] M.-R. Ra, B. Priyantha, A. Kansal, and J. Liu, "Improving energy efficiency of personal sensing applications with heterogeneous multi-processors," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, New York, NY, USA, 2012, pp. 1–10.

[41] "Smartphone And Tablet Penetration - Business Insider." [Online]. http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10. [Accessed: 13-Jun-2014].

[42] E. Basha and D. Rus, "Design of early warning flood detection systems for developing countries," in *International Conference on Information and Communication Technologies and Development, 2007. ICTD 2007*, 2007, pp. 1–10.

[43] E. A. Thomas, C. K. Barstow, G. Rosa, F. Majorin, and T. Clasen, "Use of Remotely Reporting Electronic Sensors for Assessing Use of Water Filters and Cookstoves in Rwanda," *Environ. Sci. Technol.*, vol. 47, no. 23, pp. 13602–13610, Dec. 2013.

[44] "Android Open Accessory Protocol." [Online]. http://source.android.com/accessories/protocol.html. [Accessed: 13-Jun-2014].

[45] C.-K. Hsieh, H. Falaki, N. Ramanathan, H. Tangmunarunkit, and D. Estrin, "Performance Evaluation of Android IPC for Continuous Sensing Applications," *SIGMOBILE Mob Comput Commun Rev*, vol. 16, no. 4, pp. 6–7, Feb. 2013.

[46] B. Kaufmann and L. Buechley, "Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing," in *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, New York, NY, USA, 2010, pp. 291–298.

[47] F. X. Lin, Z. Wang, R. LiKamWa, and L. Zhong, "Reflex: using low-power processors in smartphones without knowing them," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2012, pp. 13–24.

[48] "OMAP4." [Online]. http://www.ti.com/lsds/ti/omap-applications-processors/omap-4-processors-products.page?paramCriteria=no%3f. [Accessed: 13-Jun-2014].

[49] N. Villar, J. Scott, S. Hodges, K. Hammil, and C. Miller, ". NET gadgeteer: a platform for custom devices," *Pervasive Comput.*, pp. 216–233, 2012.

[50] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "PRISM: platform for remote sensing using smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, New York, NY, USA, 2010, pp. 63–76.

[51] "Phidgets - Unique and Easy to Use USB Interfaces." [Online]. http://www.phidgets.com/. [Accessed: 09-Mar-2013].

[52] "WHO | State of the world's vaccines and immunization.," *WHO*. [Online]. http://www.who.int/immunization/sowvi/en/. [Accessed: 13-Jun-2014].

[53] "WHO | E06 Temperature monitoring devices performance specifications and verification protocols," *WHO*. [Online].

http://www.who.int/immunization_standards/vaccine_quality/pqs_e6_temp_monitoring/en/. [Accessed: 13-Jun-2014].

[54] "Safe vaccine handling, cold chain and immunizations," 1998. [Online]. http://www.old.health.gov.il/download/forms/a3039_GDPv.pdf. [Accessed: 13-Jun-2014].

[55] S. Techathawat, P. Varinsathien, A. Rasdjarmrearnsook, and P. Tharmaphornpilas, "Exposure to heat and freezing in the vaccine cold chain in Thailand," *Vaccine*, vol. 25, no. 7, pp. 1328–1333, Jan. 2007.

[56] S. Setia, H. Mainzer, M. L. Washington, G. Coil, R. Snyder, and B. G. Weniger, "Frequency and causes of vaccine wastage," *Vaccine*, vol. 20, no. 7–8, pp. 1148–1156, Jan. 2002.

[57] D. M. Matthias, J. Robertson, M. M. Garrison, S. Newland, and C. Nelson, "Freezing temperatures in the vaccine cold chain: A systematic literature review," *Vaccine*, vol. 25, no. 20, pp. 3980–3986, May 2007.

[58] C. Nelson, P. Froes, A. M. V. Dyck, J. Chavarría, E. Boda, A. Coca, G. Crespo, and H. Lima, "Monitoring temperatures in the vaccine cold chain in Bolivia," *Vaccine*, vol. 25, no. 3, pp. 433–437, Jan. 2007.

[59] "IMMUNIZATIONbasics." [Online]. http://www.immunizationbasics.jsi.com/Newsletter/Archives/snapshots_volume8.htm. [Accessed: 13-Jun-2014].

[60] "GlaxoSmithKline: GSK Vaccines in 2011." [Online]. http://www.gsk.com/content/dam/gsk/globals/documents/pdf/Investors/presentations/2011/2011-03-23-martin-andrews-citi.pdf. [Accessed: 13-Jun-2014].

[61] D. Parmar, E. M. Baruwa, P. Zuber, and S. Kone, "Impact of wastage on single and multi-dose vaccine vials: implications for introducing pneumococcal vaccines in developing countries.," *Hum. Vaccin.*, vol. 6, no. 3, pp. 270–278, Mar. 2010.

[62] "Temperature Sensitivity of Vaccines," *Dept. of Immunization, Vaccines and Biologicals, World Health Organization*. [Online]. http://whqlibdoc.who.int/hq/2006/WHO_IVB_06.10_eng.pdf. [Accessed: 13-Jun-2014].

[63] "GSM/GPRS Module - SM5100B." [Online]. https://www.sparkfun.com/products/9533. [Accessed: 13-Jun-2014].

[64] "DS18B20: Programmable Resolution 1-Wire Digital Thermometer." [Online]. http://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html. [Accessed: 13-Jun-2014].

[65] "1-Wire Bus." [Online]. http://en.wikipedia.org/wiki/1-Wire. [Accessed: 13-Jun-2014].

[66] "RapidSMS: A Free and Open Source SMS Framework." [Online]. https://www.rapidsms.org/. [Accessed: 13-Jun-2014].

[67] "Django: The Web framework for perfectionists with deadlines." [Online]. Available: https://www.djangoproject.com/. [Accessed: 13-Jun-2014].

[68] "Fluke 177 True RMS Digital Multimeter." [Online]. http://en-us.fluke.com/products/digital-multimeters/fluke-177-digital-multimeter.html. [Accessed: 13-Jun-2014].

[69] "FridgeTag." [Online]. http://www.berlinger.ch/en/berlinger/main/ambient-tag/temperature-monitoring/fridge-tag.html. [Accessed: 13-Jun-2014].

[70] "SMSSync." [Online]. http://smssync.ushahidi.com/. [Accessed: 13-Jun-2014].

[71] W. Brunette, S. Sudar, N. Worden, D. Price, R. Anderson, and G. Borriello, "ODK Tables: Building Easily Customizable Information Applications on Android Devices," in *Proceedings of the 3rd ACM Symposium on Computing for Development*, New York, NY, USA, 2013, pp. 12:1–12:10.

[72] "LogTag." [Online]. http://www.logtagrecorders.com/. [Accessed: 13-Jun-2014].

[73] "Cold Cloud: Cold Chain Management Solution." [Online]. http://www.beyondwireless.co.za/. [Accessed: 13-Jun-2014].

[74] "Vaccine Guard." [Online]. http://www.vaccine-guard.com/. [Accessed: 13-Jun-2014].

[75] "ColdTrace." [Online]. http://nexleaf.org/technology/cold-chain-monitor. [Accessed: 13-Jun-2014].

[76] M. Z. Oestergaard, M. Inoue, S. Yoshida, W. R. Mahanani, F. M. Gore, S. Cousens, J. E. Lawn, C. D. Mathers, and on behalf of the United Nations Inter-agency Group for Child Mortality Estimation and the Child Health Epidemiology Reference Group, "Neonatal Mortality Levels for 193 Countries in 2009 with Trends since 1990: A Systematic Analysis of Progress, Projections, and Priorities," *PLoS Med*, vol. 8, no. 8, p. e1001080, 2011.

[77] Z. A. Bhutta, S. Cabral, C. Chan, and W. J. Keenan, "Reducing maternal, newborn, and infant mortality globally: An integrated action agenda," *Int. J. Gynecol. Obstet.*, Aug. 2012.

[78] G. Jones, R. W. Steketee, R. E. Black, Z. A. Bhutta, S. S. Morris, and Bellagio Child Survival Study Group, "How many child deaths can we prevent this year?," *Lancet*, vol. 362, no. 9377, pp. 65–71, Jul. 2003.

[79] WHO Collaborative Study Team on the Role of Breastfeeding on the Prevention of Infant Mortality, "Effect of breastfeeding on infant and child mortality due to infectious diseases in less developed countries: a pooled analysis.," *Lancet*, vol. 355, no. 9202, pp. 451–455, Feb. 2000.

[80] L. D. Arnold, "Global health policies that support the use of banked donor human milk: a human rights issue," *Int. Breastfeed. J.*, vol. 1, no. 1, p. 26, Dec. 2006.

[81] C. J. Chantry, K. Israel-Ballard, Z. Moldoveanu, J. Peerson, A. Coutsoudis, L. Sibeko, and B. Abrams, "Effect of Flash-Heat Treatment on Immunoglobulins in Breast Milk," *J. Acquir. Immune Defic. Syndr.*, vol. 51, no. 3, pp. 264–267.

[82] K. A. Israel-Ballard, B. F. Abrams, A. Coutsoudis, L. N. Sibeko, L. A. Cheryk, and C. J. Chantry, “Vitamin Content of Breast Milk From HIV-1-Infected Mothers Before and After Flash-Heat Treatment,” *J. Acquir. Immune Defic. Syndr. 1999*, vol. 48, no. 4, pp. 444–449, Aug. 2008.

[83] “DaqPRO Data Logger.” [Online]. http://www.fouriersystems.com/products/8-channel/data_logger.php. [Accessed: 13-Jun-2014].

[84] “Waterproof DS18B20 Digital temperature sensor.” [Online]. https://www.adafruit.com/products/381. [Accessed: 13-Jun-2014].

[85] “HMB Video.” [Online]. https://www.youtube.com/watch?v=yEXqJyzK0nM. [Accessed: 17-Jun-2014].

[86] M. E. Wills, V. E. Han, D. A. Harris, and J. D. Baum, “Short-time low-temperature pasteurisation of human milk,” *Early Hum. Dev.*, vol. 7, no. 1, pp. 71–80, Oct. 1982.

[87] J. Sztipanovits, “Composition of Cyber-Physical Systems,” in *Engineering of Computer-Based Systems, 2007. ECBS ’07. 14th Annual IEEE International Conference and Workshops on the*, 2007, pp. 3 –6.

[88] J. White, S. Clarke, C. Groba, B. Dougherty, C. Thompson, and D. C. Schmidt, “R&D challenges and solutions for mobile cyber-physical applications and supporting Internet services,” *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 45–56, Feb. 2010.

[89] S. de Deugd, R. Carroll, K. E. Kelly, B. Millett, and J. Ricker, “SODA: Service Oriented Device Architecture,” *Pervasive Comput. IEEE*, vol. 5, no. 3, pp. 94 –96, Sep. 2006.

[90] “Devices Profile for Web Services.” [Online]. http://schemas.xmlsoap.org/ws/2006/02/devprof/. [Accessed: 07-Sep-2012].

[91] “S90 Pasteuriser.” [Online]. http://www.sterifeed.com/s90-pasteuriser/. [Accessed: 14-Jun-2014].

[92] "WHO | Progress on sanitation and drinking-water 2010 update," *WHO*. [Online]. http://www.who.int/water_sanitation_health/publications/9789241563956/en/. [Accessed: 14-Jun-2014].

[93] A. P. Zwane and M. Kremer, "What Works in Fighting Diarrheal Diseases in Developing Countries? A Critical Review," *World Bank Res. Obs.*, vol. 22, no. 1, pp. 1–24, Mar. 2007.

[94] R. A. Hope, "Evaluating water policy scenarios against the priorities of the rural poor," *World Dev.*, vol. 34, no. 1, pp. 167–179, Jan. 2006.

[95] S. Cairncross and V. Valdmanis, "Water Supply, Sanitation, and Hygiene Promotion," in *Disease Control Priorities in Developing Countries*, 2nd ed., D. T. Jamison, J. G. Breman, A. R. Measham, G. Alleyne, M. Claeson, D. B. Evans, P. Jha, A. Mills, and P. Musgrove, Eds. Washington (DC): World Bank, 2006.

[96] "Omron HJ-720ITC Pocket Pedometer." [Online]. http://www.amazon.com/dp/B000MN92WM/?tag=googhydr-20&hvadid=30919062021&hvpos=1t1&hvexid=&hvnetw=g&hvrand=2743806253638 26265&hvpone=&hvptwo=&hvqmt=b&hvdev=c&ref=pd_sl_3y1klpxlnc_b. [Accessed: 13-Jun-2014].

[97] "DS1922L: iButton Temperature Logger." [Online]. http://www.maximintegrated.com/en/products/comms/ibutton/DS1922L.html. [Accessed: 14-Jun-2014].

[98] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data MULEs: modeling and analysis of a three-tier architecture for sparse sensor networks," *Ad Hoc Netw.*, vol. 1, no. 2–3, pp. 215–233, Sep. 2003.

[99] M. Di Francesco, S. K. Das, and G. Anastasi, "Data Collection in Wireless Sensor Networks with Mobile Elements: A Survey," *ACM Trans Sen Netw*, vol. 8, no. 1, pp. 7:1–7:31, Aug. 2011.

[100] H. E. Wang, N. C. Mann, G. Mears, K. Jacobson, and D. M. Yealy, "Out-of-hospital airway management in the United States," *Resuscitation*, vol. 82, no. 4, pp. 378–385, Apr. 2011.

[101] D. King, M. Ogilvie, M. Michailidou, G. Velmahos, H. Alam, M. deMoya, and K. Fikry, "Fifty-Four Emergent Cricothyroidotomies: Are Surgeons Reluctant Teachers?," *Scand. J. Surg.*, vol. 101, no. 1, pp. 13–15, Mar. 2012.

[102] T. M. Cook, N. Woodall, and C. Frerk, "Major complications of airway management in the UK: results of the Fourth National Audit Project of the Royal College of Anaesthetists and the Difficult Airway Society. Part 1: Anaesthesia," *Br. J. Anaesth.*, vol. 106, no. 5, pp. 617–631, May 2011.

[103] P. Howitt, A. Darzi, G.-Z. Yang, H. Ashrafian, R. Atun, J. Barlow, A. Blakemore, A. M. Bull, J. Car, L. Conteh, G. S. Cooke, N. Ford, S. A. Gregson, K. Kerr, D. King, M. Kulendran, R. A. Malkin, A. Majeed, S. Matlin, R. Merrifield, H. A. Penfold, S. D. Reid, P. C. Smith, M. M. Stevens, M. R. Templeton, C. Vincent, and E. Wilson, "Technologies for global health," *The Lancet*, vol. 380, no. 9840, pp. 507–535, Aug. 2012.

[104] "The PIH Guide to Chronic Care Integration for Endemic Non-Communicable Diseases." [Online]. http://www.pih.org/library/the-pih-guide-to-chronic-care-integration-for-endemic-non-communicable-dise. [Accessed: 14-Jun-2014].

[105] "AD8232: Single-Lead Heart Rate Monitor Analog Front End." [Online]. http://www.analog.com/en/specialty-amplifiers/instrumentation-amplifiers/ad8232/products/product.html. [Accessed: 14-Jun-2014].

[106] M. Redfield, G. Fiedler, B. Hafner, and J. Sanders, "Validating The Use Of A Kinetic Sensor For Gait Monitoring," *Rahabilitation Eng. Assist. Technol. Soc. N. Am.*, 2013.

[107] "GAITRite Systems." [Online]. http://www.gaitrite.com/. [Accessed: 14-Jun-2014].

[108] "UNICEF India - Integrated Management of Neonatal and Childhood Illnesses (IMNCI)." [Online]. http://www.unicef.org/india/health_369.htm. [Accessed: 14-Jun-2014].

[109] D. Ramachandran, M. Kam, J. Chiu, J. Canny, and J. F. Frankel, "Social Dynamics of Early Stage Co-design in Developing Regions," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2007, pp. 1087–1096.

[110] Y. Anokwa, T. N. Smyth, D. Ramachandran, J. Sherwani, Y. Schwartzman, R. Luk, M. Ho, N. Moraveji, and B. DeRenzi, "Stories from the Field: Reflections on HCI4D Experiences," *Inf. Technol. Int. Dev.*, vol. 5, no. 4, pp. pp. 101–116, Dec. 2009.

[111] N. Dell and G. Borriello, "Mobile Tools for Point-of-care Diagnostics in the Developing World," in *Proceedings of the 3rd ACM Symposium on Computing for Development*, New York, NY, USA, 2013, pp. 9:1–9:10.

[112] E. C. Larson, M. Goel, G. Boriello, S. Heltshe, M. Rosenfeld, and S. N. Patel, "SpiroSmart: Using a Microphone to Measure Lung Function on a Mobile Phone," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, New York, NY, USA, 2012, pp. 280–289.

[113] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient Backscatter: Wireless Communication out of Thin Air," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, New York, NY, USA, 2013, pp. 39–50.

[114] E. Blas and A. S. Kurup, *Equity, Social Determinants and Public Health Programmes*. World Health Organization, 2010.

[115] "AGE Reader: Non-invasive assessment of cardiovascular risk." [Online]. http://www.diagnoptics.com/en/age-reader/. [Accessed: 16-Jun-2014].