Designing Technology for Existing Infrastructure in the Developing World

Samuel R Sudar

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

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Richard Anderson, Chair

Franziska Roesner

Matt Welsh

Program Authorized to Offer Degree:
Department of Computer Science and Engineering

University of Washington

Abstract

Designing Technology for Existing Infrastructure
in the Developing World

Samuel R Sudar

Chair of the Supervisory Committee:
Professor Richard Anderson
Department of Computer Science and Engineering

Technology can be built for the developing world using existing infrastructure. Mobile and web technologies have become sufficiently ubiquitous that they can serve as powerful platforms even in low-resource settings where connectivity is intermittent or unreliable. This growth has been rapid and driven by demand from both individuals and organizations. The projects in this dissertation demonstrate ways that such existing technological infrastructure can be harnessed to augment and enhance existing workflows. I present two tools that can be deployed on this infrastructure, describe a model for matching requirements with available technical expertise, and show that threat models in data collection projects in the developing world are not fundamentally different from those in resource-rich settings. Ultimately I conclude that settings in the developing world do not present a fundamentally new class of problems. Focusing on similarities leads to recognizing the merits of existing infrastructure, building tools that fit into existing workflows, and playing to strengths on the ground.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
INTRODUCTION

Technology can be built for the developing world using existing infrastructure. Many projects aimed at low-resource settings focus on how a new piece of technology might create a novel experience or replace an existing process. This dissertation shows instead that powerful tools can be built for low-resource settings by identifying existing technological infrastructure and designing tools that refine and augment what is already in place.

The projects described here are united by the fact that they are built for commodity hardware, target existing infrastructure, and do not imagine drastically new scenarios or workflows. Instead they aim to understand and enhance existing practices. My work demonstrates that the requirements of low-resource settings do not represent a class of problems fundamentally different to those in the developed world. Indeed, the affordances of low-resource settings—things like mobile phones, intermittent connectivity, and unreliable power—are familiar to all users of technology, regardless of where we live.

1.1 Technology and Development

The internet has had a tremendous impact on global development. With the internet has come other technology, notably mobile phones. This rise of the smartphone has brought a ubiquitous computing platform to all corners of the globe. Internet access, however, has proceeded more slowly. According to the World Bank, 80% of people in the developing world have access to a mobile phone, while only 31% of those have internet access. The prevalence of these technologies is not solely constrained to the wealthy. Of the poorest fifth of the world's population, 70% have access to a mobile phone [67].

There is no question that these technologies have impacted global development. The study of these technologies and their relationship with global development is known as Information and Communication Technologies for Development (ICTD). ICTD projects include the development of technology as well as evaluating the impact of technology on communities.

Within ICTD, my work has focused on mobile technology, the internet, and workflows that use them. I do not try to answer the question of whether or not technology is a good thing for global development. Rather I take the position that this technology is having an effect whether we like it or not—better to try and have a positive impact on this trend than to try and put the genie back in the bottle.

ICTD projects take myriad forms. Some projects are rooted in business [52] while others are managed by governments [72]. Still others are maintained by non-profits (NGOs) [74] and academic researchers [45]. The fine red thread running through projects that can be classified as ICTD is that they target low-resource settings, typically in the developing world.

This dissertation focuses on commodity hardware and existing infrastructure; what is the alternative? Projects trying to increase connectivity might install dishes that provide WiFi [46] or miniature cell towers operating outside the law on frequencies that go unused in remote communities [47]. Others imagine new services that users can interact with in novel ways [83]. My work focuses on existing infrastructure and workflows, exploring how technology can augment what is already being done rather than creating brand new uses of technology.

Numerous examples of ICTD projects fitting this description can be found in the following chapters. However, I will briefly outline several projects here in order to orient those that are not intimately familiar with ICTD. These are all projects that I have been involved with. Shared traits include their emphasis on mobile tools, data collection, and workflows surrounding the deployments of technology.

1.1.1 Chimpanzee Monitoring

The Jane Goodall Institute (JGI) operates a sophisticated data collection effort in Gombe Stream National Park in Tanzania. This effort has been underway for decades, and has led to a trove of scientific data on chimpanzees as well as the state of the forest. The JGI employs locals to monitor the forest, where they identify and report signs of illegal logging, poaching, and chimp movements.

In recent years they have transitioned from paper to mobile-based monitoring. Mobile devices allow them to take pictures of a shell casing or a makeshift oven used to cook bushmeat, even providing precise GPS coordinates. The data is uploaded to a server in the evening, providing a highly up to date and detailed view of the forest.

1.1.2 Maternal Health on the Amazon

Some rural communities in the Peruvian Amazon cannot easily access the country's public health system. To help ameliorate this problem, an aid organization called the Vine Trust maintains a boat that travels up the Amazon and provides medical assistance to remote villages [84]. Every few weeks the boat moors alongside a village and villagers come aboard to receive information on health initiatives and be seen by doctors.

Mamás del Río (MDR) is a project that coordinates with this initiative to monitor maternal health in these villages. They recruited community health agents in each community, and these agents are responsible for reporting on things like pregnancies, births, and complications due to pregnancy. Many of these villages are several days by boat from the nearest population centers. This remoteness might be expected to hamper monitoring efforts.

However, Peru has a progressive policy that requires new telecommunication companies wishing to enter the country to support rural communities. As a result, despite being deep in the jungle and days away from the nearest city, the majority of these villages enjoy at least some level of connectivity. MDR leverages this by giving agents smartphones loaded with digital forms. They train the agents to collect information on the events they are interested in, collecting the data and communicating with the agents using the phones.

The critical component of this project is the focus on maternal health, not the focus on mobile technology. However, the cellular infrastructure of Peru is such that it can complement efforts like these, even in areas where communication would have been difficult only five years ago.

1.1.3 Cash Card Distribution

As one means of providing aid, the Red Cross distributes cash to victims of disasters. They have found that it is a valuable alternative to goods-based relief because it supports the local economy, removes middlemen, and empowers beneficiaries. Rather than telling someone they need a blanket and a toothbrush, it allows them to decide for themselves what best addresses their needs and then to buy it from a local seller.

As an evolution of this program, the International Federation of Red Cross and Red Crescent Societies (IFRC) wanted to pilot the distribution of pre-paid cash cards instead of hard cash. Beneficiaries receive what looks like a debit card. These cards have a set balance associated with them and can be used like a debit card until the balance is depleted. The IFRC has been an early adopter of mobile technology in relief efforts, incorporating them into their first response efforts after disasters.

They wanted to accomplish card distribution using the following workflow. Volunteers are given mobile phones loaded with a digital screening form. The volunteers take the phones into the field, finding beneficiaries and entering their data into the form on the mobile phone. They use the phone's camera to scan a bar code on a beneficiary card associating that person with a unique identifier. That evening, the data from the phones is uploaded to a central server and associated with a cash card. The next day, beneficiaries arrive at a distribution center with their bar codes. Volunteers, again using mobile phones, scan the bar codes to retrieve the beneficiary's information. Verifying that they are speaking to the right person, they see the cash card number, which has been pulled from the server, and give the card to the beneficiary.

If cost was not a concern, the IFRC could instead have deployed laptops with standalone bar code scanners rather than mobile phone camers. Indeed, when they first integrated technology into their workflows, they had a laptop equipped with bar code scanners. Mobile phones gave them the opportunity to continue to reap the benefits of digital data collection—immediate validation, minimal transcription, near real-time data, and rich types including GPS and images—at a more sustainable cost than a laptop.

1.2 Terminology and Context

The different parts of this work focus on topics sufficiently distinct that introductory material is presented at the beginning of each chapter rather than the beginning of the document. Nevertheless, some concepts and terms are sufficiently ubiquitous that they are worth introducing here.

The first is the term "developing world". When used in this document, it refers broadly to scenarios and settings where best-case technological solutions are not possible. This can include limited or reduced connectivity, limited power, and low end devices. However, it can also include personnel issues, such as low technological literacy or low capacity for training users. Many of the challenges faced by organizations under these settings could be solved by a team of programmers or by purchasing expensive proprietary software services.

Functional synonyms for "developing world" include "resource-constrained settings", "emerging markets", and "the global south". All refer broadly to the same set of conditions, and all are imperfect. The "developing world" implies a unidirectional evolution of development where some are behind others. "Resource-constrained settings" is sufficiently broad that it could include the time delay affecting astronauts on a space station, while the term in this work refers to a mostly disparate set of resources. "Emerging markets", meanwhile, implies an economic concern and could ignore relevant settings like refugee camps that exist in established European markets. "The global south" is currently in vogue as a supposedly inoffensive catch-all, but this is a spectacularly non-descriptive phrase that ignores economically developed southerly countries like Australia and northern locations like

Albania and Belarus that could fall under the development umbrella [57, 14, 64]. This document uses terms like “resource-constrained environments”, “low-resource settings”, and “developing world”, as I believe them to be the least bad of the options.

In practice, technology for the developing world as it occurs in this document is most frequently used by researchers and NGOs. These groups do not have the deep coffers of businesses that allow more expensive solutions, and they typically have small teams that include many who are not technologists.

When these organizations use technology, it is referred to as a **deployment**. A deployment can refer to a single endeavor that makes use of technology, like an effort by the organization to survey a series of households, or it can refer to how technology is used in that effort. If a research goal is to survey households, for example, and in that effort they use a mapping technology to locate houses, that application of the mapping technology can also be referred to as a deployment. In this document the latter definition, focusing on field application of a technology, is meant.

In a deployment, the **deployment architect** is the person in the organization that decides what technology will be used, and how. This person might design the workflow, e.g. mobile based with a server component, or they might design the database schema. In this document the deployment architect is whoever has the authority to make decisions regarding the technology.

An **enumerator** is a worker that uses the technology in the field. Enumerators are frequently hired by an organization, given light training, and then sent to the field to use a tool. In the field **beneficiaries** are those that provide data to enumerators.

These roles together comprise a use case in the field. For example, consider cash card distribution by the Red Cross as described above. The Red Cross has decided that they need to distribute prepaid cash cards as a form of disaster relief. They wish to devise a system in which recipients of the cards—beneficiaries—are given a unique identifier that allows them to collect a single cash card at a later date. The person who is tasked with this responsibility decides that technology can help, so they appoint a member of their organization—who becomes the deployment architect—to devise a system. They decide to use a suite of Android tools. This usage of the tools is now referred to as a deployment. Members of the local Red Cross are trained to use the tools and are on-site on to distribute cards. These workers are the enumerators, and they provide cards to beneficiaries.

1.3 Outline

The rest of this document is structured as follows.

In Chapter 2, I describe ODK Tables. Tables is a framework for writing mobile applications using web tools. Mobile phones are a critical component of developing world infrastructure, and yet writing mobile applications requires deep technical expertise. Web developers are more common than mobile developers in the developing world. This reality means that organizations cannot always employ local talent when creating mobile-based tools. Additionally, the web typically requires an internet connection, meaning that organizations requiring offline functionality cannot readily leverage the web in the field. Tables tries to bridge this gap by making mobile web technology work offline. It is a straight forward marriage of web and mobile technologies, bringing the benefits of web technology offline and increasing the utility of mobile devices.

Some scenarios in the developing world do not lean as heavily on mobile. Schools frequently provide some desktop-based infrastructure, such as semi-reliable power and a WiFi network. Offline Educational Resources (OERs) have emerged as a way to take advantage of these devices to provide educational content even without a reliable internet connection, hosting material like Khan Academy on custom pieces of hardware installed on the local network [1]. Chapter 3 describes Siskin, a tool that provides an alternative way to host OERs using only existing devices. Recognizing WiFi and the web browser as ubiquitous pieces of infrastructure, Siskin makes it possible for schools to host OERs without any additional pieces of hardware by building the capability into the browser.

Organizations using these tools operate in unconventional settings. It is well known to the security community that user expectations can differ from those of the programmers that designed the tools, creating disconnects between users and developers [2]. Chapter 4 surveys users of Open Data Kit (ODK) to explore the security models of organizations using these technologies. We hypothesized that the deployment scenarios occuring in the developing world would motivate a unique suite of security tools. In fact we found that many of the problems are the same. Encryption is desirable but usability is a barrier to adoption. Password hygiene is hard to enforce. Managing personnel is difficult. This chapter provides insight into the nature of threat models in developing world deployments and supports the idea that technical challenges in the developing world are not in a class of their own.

Technology itself is an incomplete solution. At its most effective, technology can provide tools that fit into workflows, filling gaps in function. The real difficulty is not building the tool but in figuring out what to build. Chapter 5 describes a framework for evaluating organizational workflows that employ technology. The DUCES framework provides five axes by which to judge the requirements of a particular deployment. These axes can be used to uncover complexity hiding in workflows and provide a vocabulary for talking about requirements. DUCES offers insight into how a workflow itself can be refined to create less onerous requirements of technology. Rethinking a workflow allows organizations to operate within their strengths, resisting the siren call to solve every problem with a new tool.

Finally, Chapter 6 provides concluding thoughts on technology in development settings and for the field of ICTD.

Chapter 2

ODK TABLES: MAKING MOBILE PHONES MORE USEFUL USING WEB TOOLS AND ON-DEVICE DATABASES

In this chapter I describe Open Data Kit (ODK) Tables. Mobile phones are crucial pieces of infrastructure for many organizations in the developing world, and Tables makes them more useful. Tables is a straight forward effort to marry the web with mobile technology. Several aspects of the web make it a natural choice for developing world settings. Its dynamic nature allows iteration to happen quickly, speeding up prototyping and development. Tables provides a way for web pages to behave like mobile apps, working well offline while retaining the usability of the web.

This chapter expands on brief descriptions of the tool that appeared at the ACM Symposium for Computing on Development (DEV '15) and the Workshop on Networked Systems for Developing Regions (NSDR '11) [48, 8].

2.1 Introduction

Although mobile phones are widely deployed in the developing world, their utility is hampered by the fact that app development is challenging. Writing applications that work offline and that can adapt to workflows on the ground is complex, typically requiring the expertise of mobile developers. This is true despite the fact that many mobile applications do essentially the same thing—present data to the user. This is not at its face a difficult problem. If the user can create data, the situation is slightly more complicated. The user interface (UI) must be altered to accommodate the addition of data, and network calls must be made to update the data on the server. If updates to data are also allowed, the situation is again made more complex due to interaction with a server. If the app must support disconnected operation, things become yet more complicated. A synchronization policy must be developed, and the backend must move from a simple data source to a component with logic. This in turn has implications for the data layer, which must store synchronization state, as well for the UI, which must communicate this state to the user.

Designers creating mobile applications repeatedly need to write boilerplate code to solve this same set of problems, even though it is not related to the value contribution of the app. It is a collection of such problems that they simply must solve before creating a rich, fully-featured, data-centric mobile application.

A number of tools have been created to try and provide boilerplate that solves these problems. However, the vast majority of these frameworks are aimed squarely at developers for native applications that must be used at compile-time during a conventional development cycle. There is a need for web developers and technically skilled users without the expertise of full developers to be able to create mobile applications for low-resource settings.

Projects like ODK Collect have enabled organizations to create data collection tools for Android phones without requiring developers [45, 66]. These tools have seen enormous success in maximizing human and technical capabilities in resource-constrained environments by lowering the barriers for creating mobile data collection tools [9, 41]. By simplifying the programming model to one that can be approached by users that are not developers, a larger set of users was able to take advantage of mobile-based data entry. However, one of the limitations of tools like Collect is that while data entry is simplified, interacting with that data on the mobile device is complicated [9, 41]. Creating workflows based on data that afford users the opportunity to view and synthesize information is functionality that users request but that the system was not built to support. Technically inclined users are able to collect and analyze data, but ways to richly consume and interact with that data are limited.

We present ODK Tables, an application-building framework that is a new component of the ODK suite and facilitates the creation of data-centric mobile apps by users without a strong development background. The framework has evolved from a previous description of a distributed spreadsheet shared across devices to an application building framework that facilitates mobile data management [48, 8].

Tables provides a set of abstractions and primitives that solves a set of common problems for application designers. The framework manages persistence, synchronization, and data management. Designers need only to create the UI and to declaratively define the data layer using comma-separated value (CSV) files that specify column name and type. The framework handles interactions with the persistence and network layers. The development environment becomes a browser, rather than a heavyweight development environment, simplifying configuration and the development cycle by requiring minimal configuration and eliminating compile cycles.

Application logic and the UI are written entirely using HTML and JavaScript (JS). Sensible defaults and templates are provided for common use cases, while apps remain as highly configurable as a standard web page. Using web tools to create mobile applications is similar to the approach taken by so-called "write once, deploy anywhere" frameworks that rely on the web, as well as by similar tools that do not employ the web due to performance and usability concerns. A contribution of this work is to examine the tradeoffs made when using web tools to create this class of portable applications.

The Tables framework significantly lowers the entry point to mobile app creation, making it possible for an audience wider than skilled developers or teams. In the following sections we discuss other attempts to solve these problems and why they are inappropriate for application designers with limited technical resources and know-how. We then discuss the primitives provided by Tables, followed by the architecture supporting them. We conclude by describing in more detail a number of use cases that motivated these decisions and evaluate the performance of the platform.

2.2 Related Work

Developing mobile applications is challenging. Requirements surrounding unreliable data connections and disconnected operation are made more stringent by the data-centric nature of many mobile applications that require interacting with data on a device and transferring this data to and from a server. Architectural considerations must include both consistent and efficient data storage, frugal use of the network, and the impact these decisions can have on the user experience.

Applications solve these same problems repeatedly. Several frameworks have recognized this fact and tried to simplify mobile app development by providing boilerplate that aims to solve some of these problems. However, existing tools attempt to make app creation simpler for highly-trained application designers. They are aimed at developers or institutions with the luxury of having the resources to employ teams to meet their requirements.

Yahoo's Mobius creates an abstraction of a table that spans the cloud and devices, allowing asynchronous writes and continuous queries. While useful in certain contexts, this paradigm assumes sporadic connectivity rather than extended periods of disconnected operation [20]. It was designed to be used by Yahoo's own developers in its mobile applications. Features such as continuous queries are useful in sophisticated contexts, but are not required for apps that need only to interact with and display information from a database, complemented by sporadic edits.

Izzy has been presented as a framework for building data-centric mobile applications [43]. It strives to support the creation of data-intensive applications like Dropbox and iCloud [30, 51]. Developers are presented with a compile-time API that provides abstractions surrounding data synchronization of data tables and of files. Izzy does not change the development process, but it does provide some libraries for conventional mobile development.

Twitter has developed Fabric, a suite of SDKs that intends to simplify mobile development [82]. Similar to the motivations for this work, they state that Fabric is intended to solve the problems that many developers face repeatedly. However, the abilities provided by Fabric focus on increasing the number of downloads, acquiring users, embedding Tweets, monitoring crashes, and simplifying mobile payment. While valuable for certain use cases, they do not target the basic functionality required of simple, data-centric applications we encounter in our work with public health NGOs operating in resource-constrained regions.

PhoneGap focuses on allowing a single codebase run on all mobile platforms [68]. PhoneGap applications are written in HTML/JS and run on a number of mobile platforms. This approach is similar to the approach taken by Tables; however, application designers writing PhoneGap applications still require significant technical skills. PhoneGap applications operate with a number of plugins, requiring that developers understand the ecosystem well enough to choose those that support local storage and the capability to synchronize data with whatever backend they have in place.

Other approaches have tried to push complicated application and persistence logic down to native code so that it can be written once but run on multiple mobile platforms with the help of a system-specific shim layer. Djinni was developed by Dropbox to allow a single internal team of developers to write code for all their mobile platforms [28]. The vast majority of application logic, including synchronization and interactions with the network, is written on a per-project basis using native C. Djinni is used to generate interfaces between the C layer and platform-specific native code (e.g. Java on Android, Objective-C on iOS). This approach lowers the amount of platform-specific code that must be written, but again requires the skills to write code in a specific language for each platform, as well as in native C.

2.3 ODK Tables

ODK Tables is an app-building framework that aims to simplify the process of mobile application development. It is not aimed at experienced Android developers, but instead at enabling mobile app creation by technical users that do not have a software development background. Users are expected to have some familiarity with ubiquitous tools like Microsoft Office and ideally have minimal experience making a web page. An assumption motivating this work is that individuals in resource-constrained environments are more likely to be familiar with web programming than with Android, and that this in turn will allow organizations to more easily support projects using local capacity. Previous ODK projects support the idea that targeting technically-inclined non-developers can be a powerful approach to empower users in resource-constrained environments [45, 9, 41].

Tables provides a set of abstractions and primitives that simplify the creation of data-centric apps. It identifies the most basic mobile applications as those that view and update data in a database. At its core, Tables is a framework for viewing and updating a database using HTML and JavaScript (JS). It is aimed at addressing common mobile app use cases, including those that are encountered by app designers without extensive technical expertise. It assumes all data tables undergo synchronization and thus synchronizes all tables with a server. The framework focuses on providing data-accessor abstractions rather than synchronization abstractions.

These decisions represent a tradeoff for usability at the cost of versatility. Tables supports a smaller set of use cases than do the frameworks discussed in the previous section. It does not focus on acquiring users, interfacing with native code, or providing a rich API to guide synchronization of files and data. Unlike Izzy, it does not attempt to support use cases like those required by Dropbox or iCloud apps, which require synchronization of arbitrary directories [43]. Rather it treats the primary unit of synchronization as a row in a database, and allows simple pulling and pushing of binary files. While this means Tables is not appropriate for certain types of applications, these abstractions are sufficient for a wide range of mobile applications. For example, consider an email application. Messages are represented as rows in a database. Users can view, delete, and create messages. Media attachments may be attached that are moved between the client and server.

By targeting only a subset of use cases, it is possible to greatly reduce the amount of code that must be written to create a robust mobile application. As a result, the number of changes required to customize Tables apps is much more contained than in more comprehensive frameworks. Further, the use of web tools as the basis for the UI layer encapsulates the presentation and logic from the implementation details. An app designer needs only to define their schema and they can immediately begin creating an application.

2.4 Abstractions

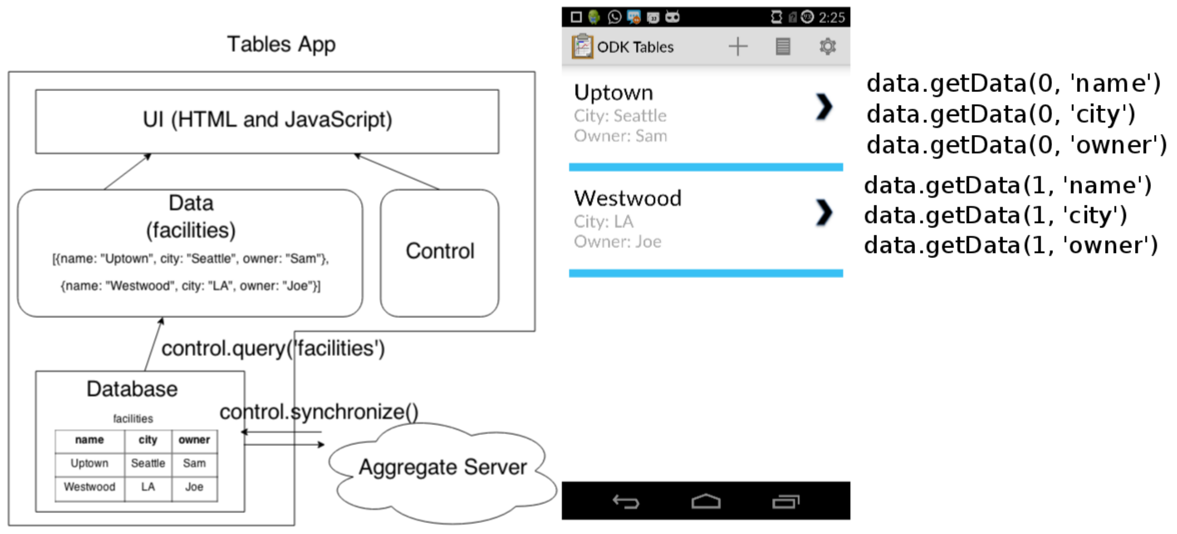
Tables provides abstractions around a number of recurring problems in application development. This section details these problems and discusses each of the abstractions Tables uses to solve them:

1. Simplified Programming Model
2. The Row as the Primitive
3. Simplified Database Management
4. Disconnected Operation
5. Synchronization
6. User Authentication
7. Data Permissions
8. Asynchronous I/O
9. Efficient Network Usage
10. Faster Development

2.4.1 Simplified Programming Model

The core aim of the Tables framework is to present application designers with a simplified programming model. Developers building applications using native Android must be familiar with the Android programming model as well as dozens to hundreds of specific APIs. The Android lifecycle, for instance, consists of a number of callback methods invoked by the system at various times and in response to different events, such as screen rotation. Pages of documentation describe the limitations and intentions of each method, and violating these guidelines results in runtime errors. Tables lowers this burden of knowledge by allowing applications to be created using web tools and a lightweight JavaScript API. Interactions with these tools can further be simplified by an application designer dashboard that can be used to template and scaffold apps for a number of basic use cases.

A local web server runs on the device, allowing application designers to interact with



```
data.getData(0, 'name')
data.getData(0, 'city')
data.getData(0, 'owner')
data.getData(1, 'name')
data.getData(1, 'city')
data.getData(1, 'owner')
```

Figure 2.1: Programming model of Tables and an example screen. This architecture is all that an app designer needs to understand to build robust mobile applications. The UI communicates with the persistence layer via *control* and *data* objects. *control.query('facilities')* returns an object that represents a database table as a list of maps. The example screen shows how two rows may be rendered using one of the provided templates, which calls *data.getData()* with the appropriate row number and column heading.

a page in much the same way they would a server. Apps consist of HTML and JS files that are rendered by the framework. Two global objects are injected into the JavaScript environment that provide context, interact with the database layer, and control the flow of the app (Figure 2.1). Illustrative subsets of these APIs are shown in Table 2.1.

Applications built this way are simpler to modify than native apps and do not need to be recompiled, opening app creation to a wider set of users. Technically inclined non-developers are able to construct apps using the app designer dashboard. Those familiar with the principles of web design can create apps as they would basic websites.

control
boolean addRow(tableId, dictOfValues)
boolean updateRow(tableId, dictOfValues, rowId)
data query(tableId[, queryClause, queryArgs])
void launchHTML(path)
boolean synchronize()

data
String getData(rowNumber, columnName)
int size()
String getTableId()
JSONObject getColumns()

Table 2.1: Selected APIs for the *control* and *data* objects available to the JavaScript environment. *control* is always available. *data* is returned by a call to *control.query()*. *data* represents rows in a database table, represented as a list of maps.

2.4.2 The Row as the Primitive

The fundamental primitive in a Tables app is the database row. An app includes a set of data tables. Querying these data tables returns a list of maps, where each map represents a row in the table.

Updates to the database are also made at the row level. Inserts result in the creation of a new row, edits of a single field flag the row as dirty. Synchronization is conceptualized as the process of updating the set of rows on the server and client to be identical. Similarly, permissions are granted to users at the row level rather than by granting different users the ability to only interact with a subset of columns in a row.

Apps are structured around interactions with these rows. Each data table canonically includes two files providing two views—a list, providing an overview of multiple rows in the table, and a detail view, providing a detailed view of a single record. These files are defined using HTML and JS, allowing navigation between views to occur via links. If an application consists of data tables defining facilities and patients, for example, the detail view of a facility might contain a link to a list of patients at the facility, facilitating joins between data tables.

2.4.3 Simplified Database Management

Apps that support disconnected operation must persist data using local storage. This is often accomplished with a database, such as the SQLite implementation that ships with native Android, but can also be accomplished using flat files. Boilerplate code providing data accessor abstractions onto these persistence layers is required of all mobile applications.

In Tables apps, the database initialization process is streamlined. Application designers declaratively specify their column identifiers and column types using CSV files. Schema creation occurs during app initialization. No explicit database definition is required, lowering the burden on the application designer.

JS objects are injected into the environment by the Android component of the app. One of these objects provides access to the database. Queries can be performed using a simplified syntax for basic cases and also support the full power of SQL. Filters can also be realized directly in JavaScript, making knowledge of SQL required only when operations on large datasets require improved performance.

Table 2.2 summarizes the approach Tables takes to database management and compares it with the approach required of a canonical native Android app. App designers using Tables need to be familiar with only two APIs and the declarative identifier/type schema definition syntax to have access to a full-featured database with accessor abstractions.

2.4.4 Disconnected Operation

Robust mobile apps are expected to function while not connected to the Internet. Disconnected operation is especially important in resource-constrained environments where fast,

Concept	**ODK Tables**	**NativeAndroid**
Schema Definition	Excel or CSV file, configured automatically	Saved as native SQL strings in Java code, executed in a callback in the Java environment
Data Accessors	Map of column name to value	Coded by hand
Joins	SQL exposed or in memory	SQL exposed or in memory
Languages	JavaScript, HTML	Java, SQL, XML
APIS	*control*, *data*	SQLiteDatabaseHelper, SQLiteDatabase, Cursor, ContentValues, ContentProvider, UriMatcher, ContentResolver

Table 2.2: Comparison of database management approaches required by Tables and native Android.

reliable connections are uncommon or unavailable at an affordable cost.

Tables apps function completely offline. The presentation layer uses HTML and JavaScript files stored on the device, serving and rendering them to create the user interface. Dynamic data is stored in a local database. All CRUD operations are supported while disconnected. Creates, updates, and deletes modify the database and are resolved using a synchronization protocol during periods of connectivity.

The synchronization protocol does not make assumptions about the periodicity of connectivity, so Tables apps can function for days without synchronizing. Data can be imported and exported using CSV files for use cases where connecting the device itself to a network is not possible and files are able to be physically transported.

2.4.5 Synchronization

Allowing create, update, and delete operations while disconnected is necessary to permit full disconnected operation. However, it necessitates a synchronization protocol. The Tables framework provides a robust synchronization protocol so that app designers do not have to implement this functionality. At a high level, the Tables synchronization protocol has been outlined elsewhere [8]. However, additional refinements have been made. These are outlined in the architecture section. The essential abstraction surrounding synchronization within the Tables framework is that the protocol operates at the level of the database row. Individual rows are synchronized with the server, which aligns with the Tables data model of a row as the data primitive.

The default server provided for immediate deployment is an instance of ODK Aggregate [45]. Aggregate can be deployed on MySQL, PostgreSQL, or the Google App Engine Datastore, and can be installed with a one click installer. However, the Tables framework is decoupled from Aggregate and can be deployed with any server that implements the synchronization protocol.

This model carries implications for the app designer. Synchronization can be triggered with a call to the injected *control* object, allowing designers to customize the synchronization experience presented to the user. The app can also be configured to sync by default during times of connectivity, further reducing the burden on the designer. Conflicts can be presented to the user via a default UI provided by Tables, at which point they are resolved by the user, within the JavaScript layer. Table 2.3 outlines the concepts surrounding synchronization and compares how an app designer must address them while implementing an app using Tables versus native Android. Here again Tables gives up some versatility, such as the ability to synchronize only certain tables, in the name of usability. App designers need only to make a single call to *control.synchronize()* to synchronize app data with a server, requiring significantly less knowledge and configuration than native Android.

Concept	ODK Tables	NativeAndroid
Determine Connectivity	Handled by framework	Get the network status, register a receiver for network changed events
Trigger Sync	Invoked via JavaScript or automatically by framework	Invoke the sync logic at the appropriate time based on connectivity
Protocol	Handled by framework	Devised by developer
Server	ODK Aggregate by default, deployable on MySQL, PostreSQL, and Google Datastore	Implemented by developer or configured to speak to a known server, as well as speak the protocol
APIS	*control*	ConnectivityManager, BroadcastReceiver, SyncAdapter, NetworkInfo, HTTP clients

Table 2.3: Comparison of synchronization approaches required by Tables and native Android.

2.4.6 User Authentication

Many applications employ a user-based model that require users to authenticate before accessing resources. Applications such as Facebook and Dropbox implement their own account types and authentication machinery. However, properly storing this information is complicated, and the burden is high on developers to maintain adequate levels of security [32]. Identity as a service has emerged as a solution to this problem [7]. Rather than re-implementing best practice protocols like OAuth2, developers can rely on identity and authentication APIs that interact with machinery implemented by companies such as Google and Facebook. The onus of maintaining adequate security is on these identity providers, who are better equipped to handle the problem.

However, interfacing with these APIs remains a challenge. As many as 59.7% of the most popular OAuth-capable apps on Google Play have incorrectly implemented the protocol and are vulnerable to attacks [15]. This is in part because of high demands on developers. Both the client and server must be equipped to interact with the protocols, which requires knowledge of libraries in client and server languages. Identity tokens must not be exposed in URLs, which are not encrypted by SSL. HTTPS must be used during communication, yet obtaining SSL certificates can be expensive and a significant barrier to developers, especially in resource-constrained environments.

The Tables framework solves these problems by interacting with the Google identity APIs. The decision to use Google was made due to the fact that Tables is an Android-based framework and most users will likely already have Google accounts, but the tool could be extended to employ additional identity providers. By default ODK Aggregate is deployed to Google App Engine, which provides a free SSL certificate. App designers need only use the Google dashboard to associate Google Apps accounts with their application in order to ensure that all communication is secure and authenticated.

2.4.7 Data Permissions

In many cases users need access only to a subset of data on the server. For example, a doctor should only have access to the records of patients under their care, not all the patients on record at the clinic. Tables handles this using a row-based permissions system on the server. Users are associated with rows on the ODK Aggregate dashboard. Data is filtered during synchronization. When a device syncs, only the rows that user is permitted to see are pulled to the device.

In some applications, all users see all data. For example, a simple weather application might present all data to all users. In other cases data is more sensitive and should be isolated between users. In the doctor example above, for instance, two doctors sharing a device might receive different datasets from the server during synchronization. Both datasets are now present on the device and should be filtered during user interactions. The Tables framework does not apply a default option to filter data at the application layer. However, data can be filtered both by the database layer and in memory, affording application designers the ability to add this functionality with a small piece of application logic.

2.4.8 Asynchronous I/O

Mobile best practice dictates that data should be loaded asynchronously. This allows the UI to remain responsive while potentially long operations complete on a separate thread. Asynchronous loading is especially important on mobile devices, where computational resources are more modest than on desktop applications.

Asynchronous loading is a ubiquitous paradigm in web development, where all I/O is assumed to be happening over the network and could take arbitrary amounts of time. Asynchronous calls are a natural construct in JavaScript, as functions can be passed as first-class objects. Rather than implementing a system of callbacks, coupling pieces of code together, functions can be passed to AJAX requests and be implemented with no additional overhead on the part of the developer.

The Android framework provides developers with some classes that facilitate asynchronous loading, but these classes introduce complexity and require a significant amount of boilerplate simply to gain access to data. Naively, this might include wrapping a thread with an AsyncTask, associating the thread with the UI element in a way that is tolerant of device rotation, and updating the UI on load completion. A slightly less naive approach could include passing a Cursor to a number of UI elements Android provides, but this requires knowledge of APIs like Cursor and ContentProvider, and it couples the UI to the data layer. The Android best practice employs Loaders, which are capable of decoupling the data layer and UI layer, but are nontrivial to implement correctly, and require knowledge of an even more complex API system including Loader, AsyncTaskLoader, and the LoaderManager.

Tables leverages the asynchronous-first philosophy of JavaScript and AJAX to avoid this complexity. By placing the UI and application logic in JavaScript, the Tables framework allows developers to more readily load data asynchronously. Apps written using JavaScript are able to more easily follow the mandate that I/O should happen without freezing the UI. Synchronous loads are still available to the developer, if required, or if performance implications are negligible.

2.4.9 Efficient Network Usage

Disconnected operation is an integral feature of mobile apps. While connected, however, they are faced with the additional expectation that they are efficient with regards to data usage. Mobile devices are frequently moving between networks and do not enjoy the stable, strong connections common to desktop devices. These mobile networks are often expensive and impose quota limits.

Tables data synchronization improves on this model. Dynamic data is synchronized at the granularity of a row, ensuring only the delta between data is synchronized. Conventional Android applications are capable of replicating this behavior with custom synchronization protocols, but as discussed above, this is a significant burden on the implementer.

System and software updates impose a heavy burden on networks in resource-constrained environments [88]. Standard Android applications are distributed as compressed files. Application updates occur as a new compressed binary file that is downloaded and installed to the device. Delta changes are not computed. Consequently even an update to a text file correcting a spelling mistake can result in a large binary file being downloaded to the device. The Tables app model permits more efficient file-level synchronization. Only files that have changed are pushed from the server to the device. Applications can thus be updated at a much lower network cost than using the default distribution mechanism.

2.4.10 Faster Development

Iteration is a standard process in application development. It is especially important in designing user interfaces, where application design often progresses incrementally toward a target mockup or to meet the specifications of another party. Iterations during Tables development proceed much faster than is possible with native Android development due to the fact that Tables employs runtime rather than compiled languages.

Native Android user interfaces may be defined in Java, in XML, or in a combination of the two. Previews can be generated by Android developer tools on those views that are defined solely in XML. However, a UI that depends on runtime, including dynamically composed or modified elements and content, cannot be previewed. Iterations in Tables are substantially faster due to its use of runtime web files to define the user interface and logic. The HTML and CSS defining the UI can be modified while the application itself is running, allowing immediate feedback and updates. This drastically lowers the time between iterations. Iterating using native Android requires making a change, compiling, and deploying to a device or emulator.

2.5 Architecture

A Tables app has two components. The first is the framework itself, which exists as a single binary Android installation. This needs to be installed, but never modified, by an app designer. The second component is a collection of files that define the Tables app. This includes HTML and JavaScript files defining the UI, media attachments, and configuration files. These files are isolated from each other, allowing a single Tables install to serve a number of different apps on a single device.

2.5.1 Framework APK

The Tables framework is an Android app installed as an Android application package, or APK. This is the native component of the application that serves as the interface between the user application and the device itself. The APK handles all interactions with the Android and Java frameworks. Such interactions include reading files from disk, calling the operating system to launch intents to start other apps, and managing a SQLite database. Most importantly, the APK also runs a lightweight web server capable of serving files from within the app's directory space.

2.5.2 UI Layer

The UI layer of a Tables app is written in HTML and JavaScript. Each screen in a Tables app is merely a customized web page rendered by the Tables APK. HTML files are presented in an Android WebView, which renders HTML and executes JavaScript. All standard web functionality is supported in this WebView. Navigation can occur between additional web pages stored locally on the device, and standard JavaScript libraries and templates can be used. These files are served by the Tables APK, which runs a lightweight web server. It is possible to create Tables apps that depend on files available only via the network. However, apps that intend to support disconnected operation must be designed accordingly and not rely on network availability.

2.5.3 Interface between Native and Web

Communication between the native and web components of a Tables app proceeds in two ways. The first is simply via the web server, which runs as Android code at localhost and vends files to the WebView that is rendering the UI. More powerful interactions are made possible by the second method, which involves injecting objects into the JavaScript context running in the Android WebView. These injected objects are bound to global identifiers in the JavaScript environment.

There are two classes of such objects. The first, *control*, allows callers to perform actions that are not associated with a particular data set. This includes querying the database, retrieving information about the context under which the application is running, and adding or editing rows in the database. *control* is always present on any page that is rendered using Tables.

The second class of object, *data*, is an interface for interacting with a set of rows in a data table. *data* objects are created by calls to the *control.query()* family of methods (shown in Table 2.1), and app designers are able to access multiple *data* instances in a single page.

A row is represented as a map. The key set of the map is the set of columns in the database, while the values are the contents of that particular column. A *data* object can thus be thought of as a means of interacting with a data table.

Marshaling between the Java and JavaScript layers is performed by the Android platform itself. The complete behavior of this process is not specified as part of any public API that is known to the authors. However, experiments suggest that only JavaScript primitives and String objects are passed between the Java and JavaScript layers. This limitation means that, for example, a JavaScript Object Notation (JSON) object can only be passed from Java to JavaScript as a String, requiring the content to be parsed before being used as a JSON object.

2.5.4 Synchronization

Synchronization of a Tables app occurs at a file level and a data level. The file level includes the web files that define the UI and a set of configuration files defining the database schema. Configuration files include database definitions in the form of CSV files, as well as file manifests guiding app initialization. For example, a manifest might include the path of a file that contains data with which to prepopulate the local database. These files can be created by hand or with the application designer dashboard. Web files include the HTML and JavaScript files that are rendered to create the UI of a Tables app. The local directory structure is mirrored on the server, which represents the ground truth of the app installation. Changes to files on the server are pulled to the phone during synchronization, allowing application updates without requiring a binary install. App synchronization begins by retrieving a file manifest from the server. This manifest includes file name, the hash of the file, and a download URL. This manifest is compared with the state on the phone. Files not present or with hashes differing from the server files are downloaded. Files not present on the server but present locally that are not media attachments to data table rows are deleted.

Data synchronization occurs at a row level. The protocol has been previously described but is summarized here [8]. Only ground truth rows are allowed on the server. Synchronization state (i.e. clean or dirty) for a given row is stored in the client database along with a row content entity tag (etag). Client-side creates are immediately accepted on the server. Rows that have been edited by the client are compared to the current ground truth on the server. If the content etags match, indicating that the row has not been changed on the server, edits are pushed to the server. If the content etags differ, meaning that the client and server rows have diverged, the row enters a conflict state and the client-side data is not pushed to the server. Conflicts must be resolved locally, either by manual intervention on the part of the user or through an automated resolution strategy specified in JavaScript and included in the configuration files.

A strength of this synchronization policy is that it supports incremental progress on both challenged and robust networks. On a robust network, a 30-column, one hundred row table without media attachments takes approximately three seconds to download from a server and five seconds to upload. When each row contains a 1.2MB JPG file, the 100 rows and the media files take approximately 2.5 minutes to download and 4.5 minutes to upload.

2.6 Use Cases

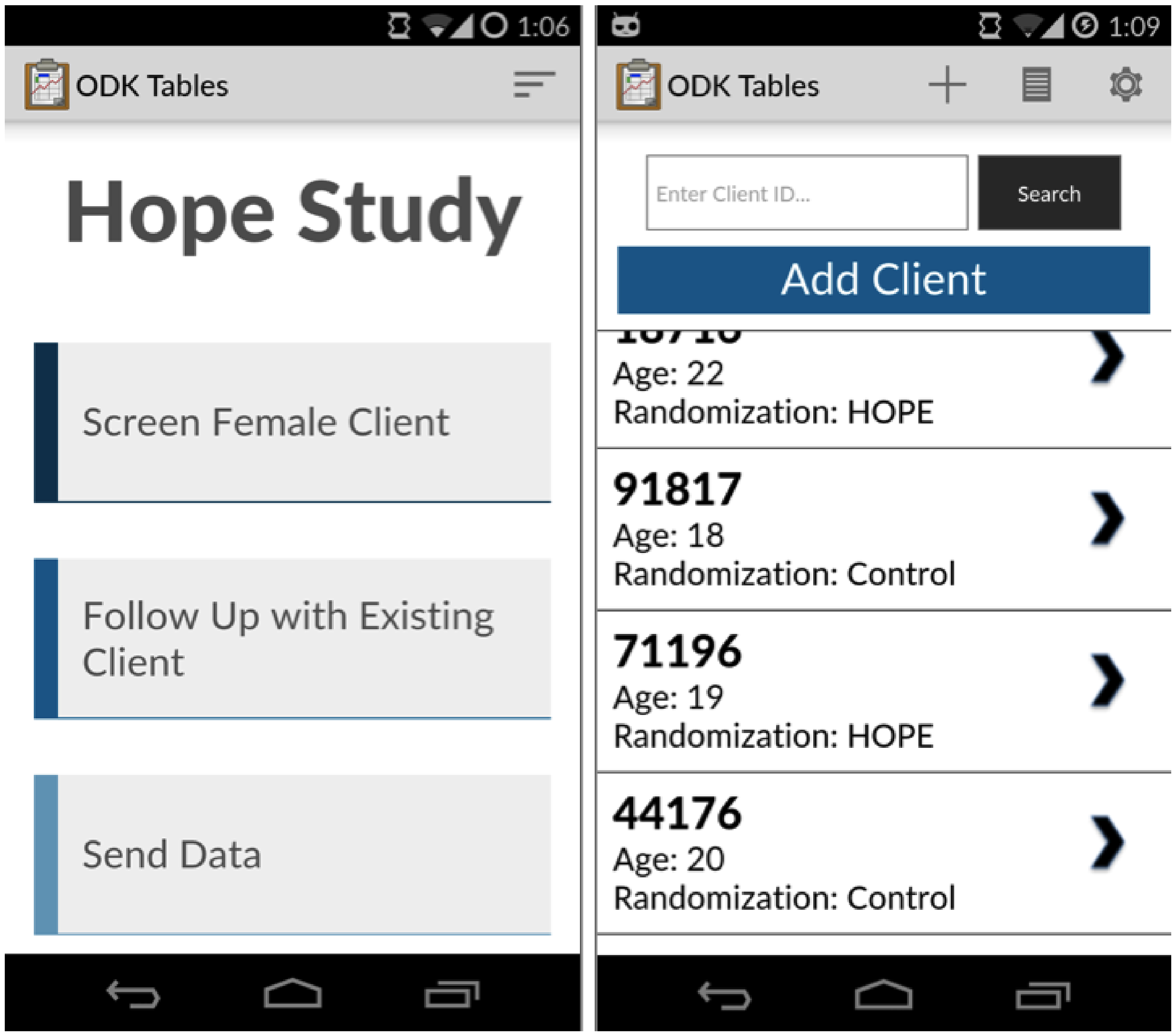
The functionality provided by Tables allows mobile applications to be designed for organizations that normally do not have sufficient resources to design a robust mobile app themselves. This section details three use cases of Tables, each of which highlights different functionality, and discusses how the framework facilitated their creation. Several of these deployments are also discussed in Chapter 5. The treatment here focuses on implementation details on top of the Tables framework, while the later discussion centers on the workflows into which these tools are integrated.

2.6.1 Hope Study

The Hope Study is a longitudinal study tracking HIV-discordant couples in Kenya. It includes a Tables app component that has been running continuously for over 15 months [78, 77]. Study participants are administered a survey at various time points over the course of a year. The surveys are completed on a smartphone. Enumerators come to the study center in the morning, retrieve the list of participants they need to visit that day, and travel to the field to find the patients. The participants live in makeshift housing in areas that do not have connectivity, so interactions with the app must not depend on a data connection. Once encountering a participant, the enumerators complete a survey for the participant.

A Tables app was created to simplify this workflow. When opening the app, enumerators are presented with three options: they can screen a new participant, follow up with an existing participant, or submit the data to the ODK Aggregate server providing the datastore for the deployment (Figure 2.2). If enumerators elect to follow up with a patient, they see the list of patients for which they are responsible (Figure 2.2). They identify patients by study code. Searching for these codes is facilitated by a search box, which has been implemented in JavaScript. Selecting a patient launches a detail view that displays the subset of information in the database that was deemed pertinent by the designers. This includes anonymizing patient identifiers and whether they are in the control or intervention arm. Enumerators are also presented with buttons that correspond to the available forms pertinent to this patient. Selecting a button will launch ODK Collect (one of the surveying tools available in the ODK tool suite) to fill out the particular survey instrument.

The basic functionality of the app is to guide user interaction, view a database, and associate forms with users. All of this would be possible using a native Android app, but Tables simplified the process. The database schema is instantiated from the survey forms, at which point the framework handles all data management. The pairing of forms with users and the creation of the UI employs a total of six HTML/JS files—one HTML and one JS file for each the home screen, list, and detail views, none longer than 132 lines. An Android app accomplishing the same goals would require at least a Fragment or Activity for each screen, a ListAdapter, a SQL definition of the database, network interactions to push data, and knowledge of intent launching to call the data entry software. While direct comparisons are difficult, the Hope Study app was written entirely by an undergraduate student with exposure to JavaScript only through an introductory web programming class.



Age: 22
Randomization: HOPE

91817
Age: 18
Randomization: Control

71196
Age: 19
Randomization: HOPE

44176
Age: 20
Randomization: Control

Figure 2.2: Home screen and patient list view for the Hope Study app. These views are defined using HTML and JavaScript.

Changes to the UI suggested by the team were able to be prototyped in real time, given the runtime nature of the HTML layer, which can be updated dynamically. The app has been in successful use for over three years in western Kenya.

2.6.2 Disconnected Branch Assignment

A second study is currently being piloted using the Tables framework. The study goals are similar to those of the Hope Study in that participants need to be tracked over a period of a year. This study is focused on locating patients that are not compliant with their antiretroviral medication regimen. The protocol entails a thorough tracking and follow-up process on the part of the enumerators. The study organizers expect this process to require several days in the field without connectivity. Within these bounds, the study benefits from a Tables app for the same reasons as the Hope Study, in that it natively supports extended disconnected operation.

The study differs from the Hope Study in that it has multiple arms beyond simply an intervention and control. Further, participants advance from one randomly assigned arm to another as they progress through the study. For example, after a patient exits arm A, they may be assigned to arm B or C. The protocol specifies that this randomization must occur in the field at the time that the enumerator has located the patient, and a flow-chart style logic is followed to select the correct form based on randomization arm and patient information. This is a simple task using the Tables framework. The enumerator selects a randomization button, and the randomization is performed in JavaScript according to the study protocol constraints. This variety of lightweight customization is made trivial by the Tables framework, which can be modified to include a randomization feature while not requiring a development environment beyond a browser.

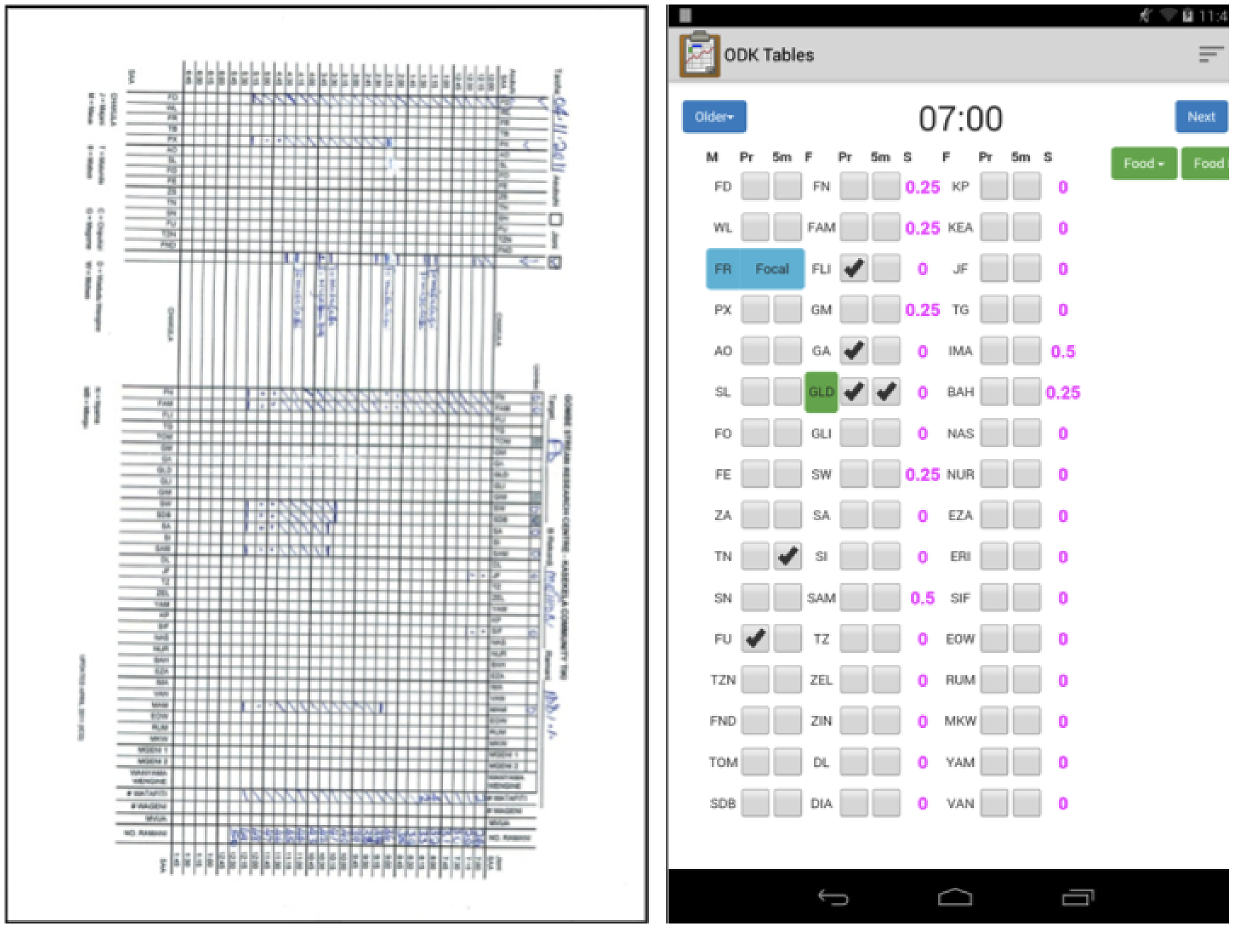
2.6.3 Unconventional UI

The two previous studies demonstrate how Tables can be used to develop list and detail views onto database tables with minimal configuration. However, the abstractions provided by the framework are sufficiently powerful to support drastically different models using the same paradigm. Another study is being piloted that uses a Tables app as an alternative to form-based data entry. In this scenario, park rangers follow troupes of chimpanzees through the jungle with a clipboard and a paper form.

The paper form is organized into a grid format with names of chimpanzees and boxes indicating presence or absence, proximity to the group leader, and estrus state (Figure 2.3). A new form is completed every 15 minutes, and at the end of the day the information is transcribed and added to a database. Using paper, there is no way to enforce internal consistency or quality of the data. However, the structure of the form as a grid prevents it from being converted to any of the offline digital data entry tools employed by the organization. Maintaining the format is of high importance to the organizers, as it has been used for a number of years and the rangers are comfortable the form.

Conceptualizing the workflow as a series of web pages allows the form to be converted to a Tables app. The grid is created using an HTML table. Each table row is represented by a row in the database associating the current time point and the chimp ID. Edits to the data update the corresponding row in the database via a call to *control.updateRow()*. Internal logic is implemented once and applied to each row. Additional value is added by pre-populating information from the previous time point, lowering the burden on the ranger by reducing the duplicate effort of re-entering data every 15 minutes.

An example of the paper form is shown in Figure 2.3. While more complicated than the previous examples, this app is also able to written with a much smaller codebase than would be required using native Android. The database is again defined declaratively using CSV files. The page consists of a single page of HTML and roughly 1200 lines of JavaScript. While this approach requires some basic programming knowledge, to do the same using Android would require extensive knowledge of the Android persistence APIs, familiarity with the Android GridLayout, and knowledge of the Android activity lifecycle and APIs. A Tables-based approach creates a more encapsulated and manageable app, where changes are more contained.



M	Pr	5m	F	Pr	5m	S	F	Pr	5m	S
FD			FN			0.25	KP			0
WL			FAM			0.25	KEA			0
FR	Focal		FLI	✓		0	JF			0
PX			GM			0.25	TG			0
AO			GA	✓		0	IMA			0.5
SL			GLD	✓	✓	0	BAH			0.25
FO			GLI			0	NAS			0
FE			SW			0.25	NUR			0
ZA			SA			0	EZA			0
TN		✓	SI			0	ERI			0
SN			SAM			0.5	SIF			0
FU	✓		TZ			0	EOW			0
TZN			ZEL			0	RUM			0
FND			ZIN			0	MKW			0
TOM			DL			0	YAM			0
SDB			DIA			0	VAN			0

Figure 2.3: Creating data-centric apps with unconventional UI is simplified by Tables and the use of web tools. A grid-based paper form is replaced with a Tables app that simplifies data entry.

2.7 Metrics

In this section, we compare the performance of Tables with that of native Android. This data is not intended to provide an exhaustive comparison of web browser and Android performance, nor to benchmark Android versions and devices. Instead this analysis makes a pragmatic comparison of Android-based apps and HTML/JS-based apps. Tradeoffs are made when selecting frameworks, and this section evaluates the performance implications of those tradeoffs. In other words, if a developer chooses to design a mobile application using web tools, what rough performance can they expect compared to a native app?

An app written using the Tables framework adopts a very different architecture than a standard Android app. All rendering is performed in the browser, objects must be injected into the environment, and data values must be marshaled between the Android and web layers. These extra steps create opportunities to incur performance penalties when compared to native Android applications. These performance penalties have been cited as a motivating factor of using native C code over a web-based approach for Djinni, for example [31].

However, there are reasons for optimism when considering the extent of these penalties. First, the browser has been the focus of extensive optimization efforts, particularly in the execution of JavaScript. Since Android 4.4, the WebView object responsible for rendering HTML/JS is backed by the same Chromium project supporting Google Chrome, ensuring that it benefits from these optimizations. Second, the WebView relies on native C code, which can perform differently than pure Java-based Android code. The following sections evaluate the penalty paid by apps regarding the injection of objects, the marshaling of data into the web layer, and of CPU usage during these operations. The first two experiments were run on a Samsung Galaxy S5, while the CPU usage experiment was run on a Nexus 7 tablet. Both devices were running Android 4.4.2.

2.7.1 Java to JavaScript Injection

Access to the persistence layer is provided to Tables apps by injecting Java objects into a WebView. These object are made available globally to JavaScript executing in that environment. Queries returning references to objects containing 10, 100, 1 thousand (1k), and 10 thousand (10k) 30-column rows were performed 250 times. The duration of a synchronous call to query the database, bring the result set into memory as a list of maps, and return an object with references to the results was measured in Android code and JavaScript. The steps were identical, except that the JavaScript calls were initiated from JavaScript and injected back into the environment, while the Android calls were initiated from an Activity.

The distribution of duration results of 250 queries are shown in Figure 2.4.

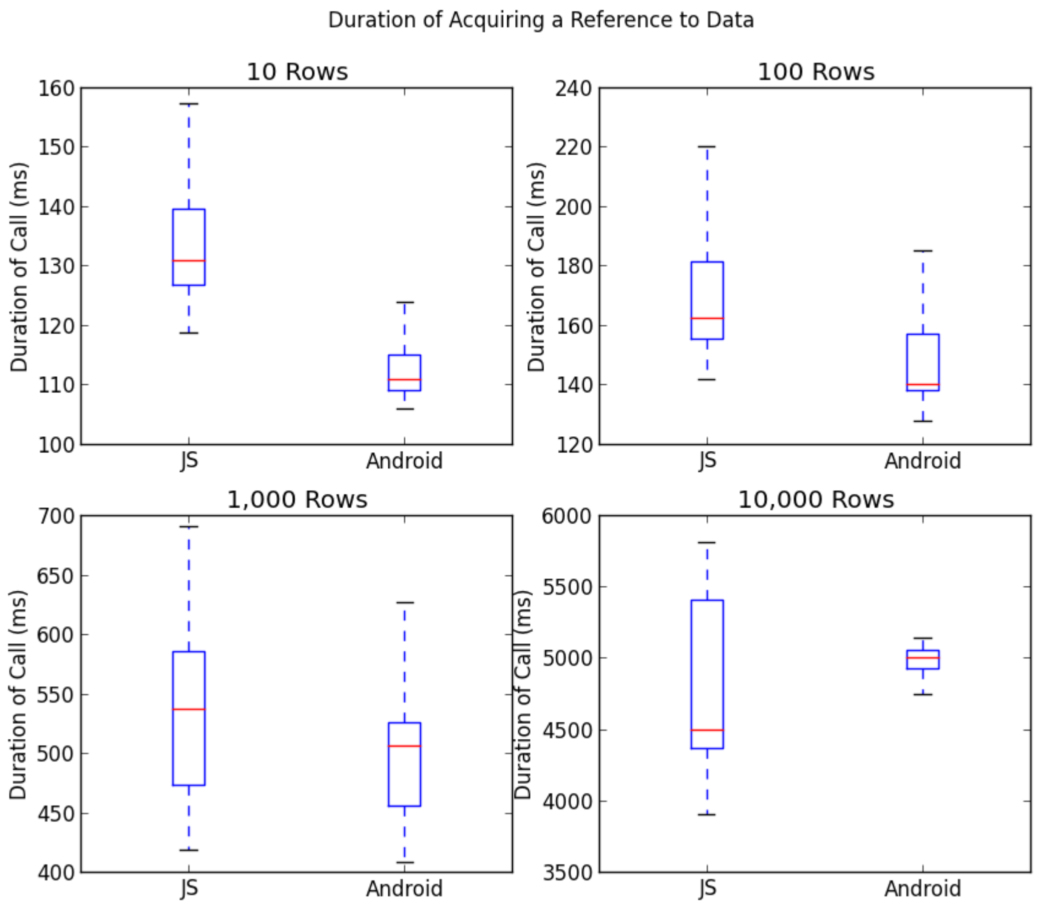


Figure 2.4: Distribution of time to acquire a reference to *data* objects in JavaScript (JS) and native Android over 250 calls. Outliers have been removed.

These queries prepare the data in memory but do not return the data itself. Rather the call returns an object that contains a list of maps representing data, which is the implementation provided by the Tables API.

Call duration between Java and JavaScript is comparable for queries of all four sizes. The Java code is slightly faster for the 10, 100, and 1k row queries, but slower for the 10k row queries. The JavaScript code requires the same steps as the Java code but with the added work of injecting the reference into the JavaScript environment. Based purely on the number of operations, it is surprising that fewer operations take longer to complete and that the JavaScript performs better than Java code. Further, the difference for the queries returning fewer than 10 thousand rows does not differ by a standard amount. This suggests that there is a cost beyond simply injecting the object.

Figure 2.5 shows the duration of the individual calls over the 250 queries. The first call is the most expensive for all cases except the JavaScript query for 10k rows, suggesting that subsequent queries benefit from caching performed by the OS. The JavaScript queries are subject to more variation than the native calls, especially in the 10k row case. Similar spikes in call time are seen for the 10, 100, and 1k row cases. Calls in JavaScript for the 10k row case, however, exhibit a striking periodicity that is absent in the Java code.

It might be expected that the JavaScript code would be slower than the Java code by a constant set amount—the price of injection. However, the data show that this simple model does not reflect true call durations, likely due to garbage collection. The Android log during the Java runs indicated that on average approximately 104,000KB of memory were available and being freed during the run. JavaScript runs, meanwhile, had approximately 62,000KB of memory available. These differences are likely due to the context of the calls. The Android code is Java executing entirely in the JVM. The JavaScript code, meanwhile, is running partially as native C code, which the system may be partitioning differently. While less efficient, the delay is on the order of several hundred milliseconds rather than seconds. The mean duration for these calls is shown in Table 2.4.

The bulk of the time querying for 10 rows was spent retrieving the schema information from the database about the table being queried. This is a necessary step for interactions with the persistence layer due to the fact that Tables databases are dynamically generated and configured. As the number of rows increases, retrieving schema information becomes less significant, and reading the rows from disk begins to dominate. This issue is studied in more depth in the discussion section.

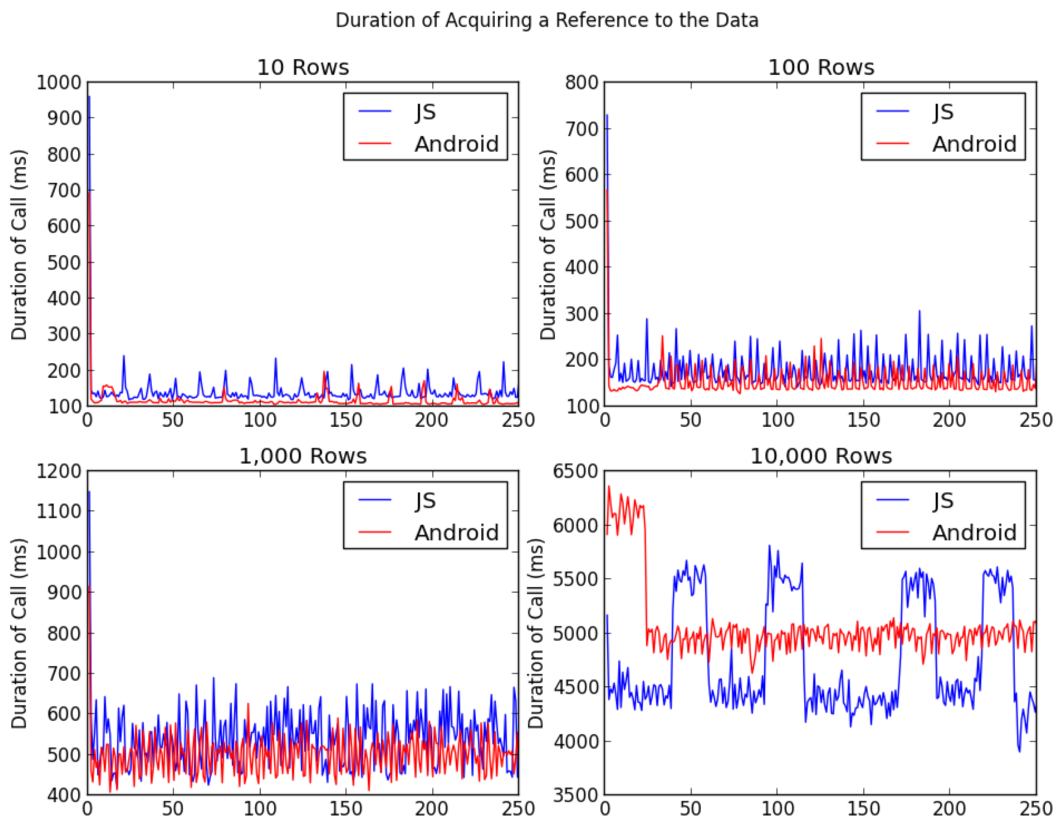


Figure 2.5: Duration of 250 calls to acquire a reference to a *data* object backing 10, 100, 1 thousand, and 10 thousand rows in Android and JavaScript.

2.7.2 Marshaling Data to JavaScript

Caching the data and making it available to applications is a useful metric when considering the time synchronous queries and asynchronous callbacks take to complete, but the cost of marshaling the data itself between the two boundaries also has implications for app designers, as they need to use the data, not merely prepare it in memory.

Again queries for 10, 100, 1k, and 10k 30-column rows were performed 250 times. Rather than simply injecting the *data* object, these queries returned a stringified version of the data.

	Reference (ms)		String (ms)	
Rows	**JS**	**Android**	**JS**	**Android**
10	140.89	117.58	146.93	120.04
100	177.83	153.24	216.65	187.01
1k	540.19	500.63	986.05	919.88
10k	4,763.44	5,075.04	9,981.34	8,590.16

Table 2.4: Mean duration over 250 runs to acquire a reference to the data and a string representation of the data in Java and JavaScript.

The *data* objects in the Java layer stored all their data as a string, and the JavaScript layer requested this string. The string representation consisted of approximately 8k, 80k, 800k, and 8M characters, respectively. In contrast to the queries comparing the cost of injection, these queries measure the cost of passing the data itself from the Java to the JavaScript. The distributions of the results are shown in Figure 2.6.

In this case, the JavaScript queries require more time for each query size. Median call durations were within 150ms for all queries except the 10k row table, where the median JavaScript call duration was slower by approximately one second. Figure 2.7 shows the duration of individual calls during these queries. Similar periodicity is shown in all cases except the 10k row queries, where the JavaScript exhibits dramatic spikes in duration due to garbage collection.

Time differences between the Java and JavaScript do not scale directly with the number of characters being passed to the JavaScript. Each query is larger than the previous query by approximately a factor of 10 (although not precisely due to the manner in which the data was generated), and yet the difference in durations does not scale by approximately 10. As with the previous queries, this is likely due to garbage collection, which is handled differently under the two conditions. It also is not clear that marshaling between the layers can be expected to scale directly with the size of the marshaled string, as marshaling is

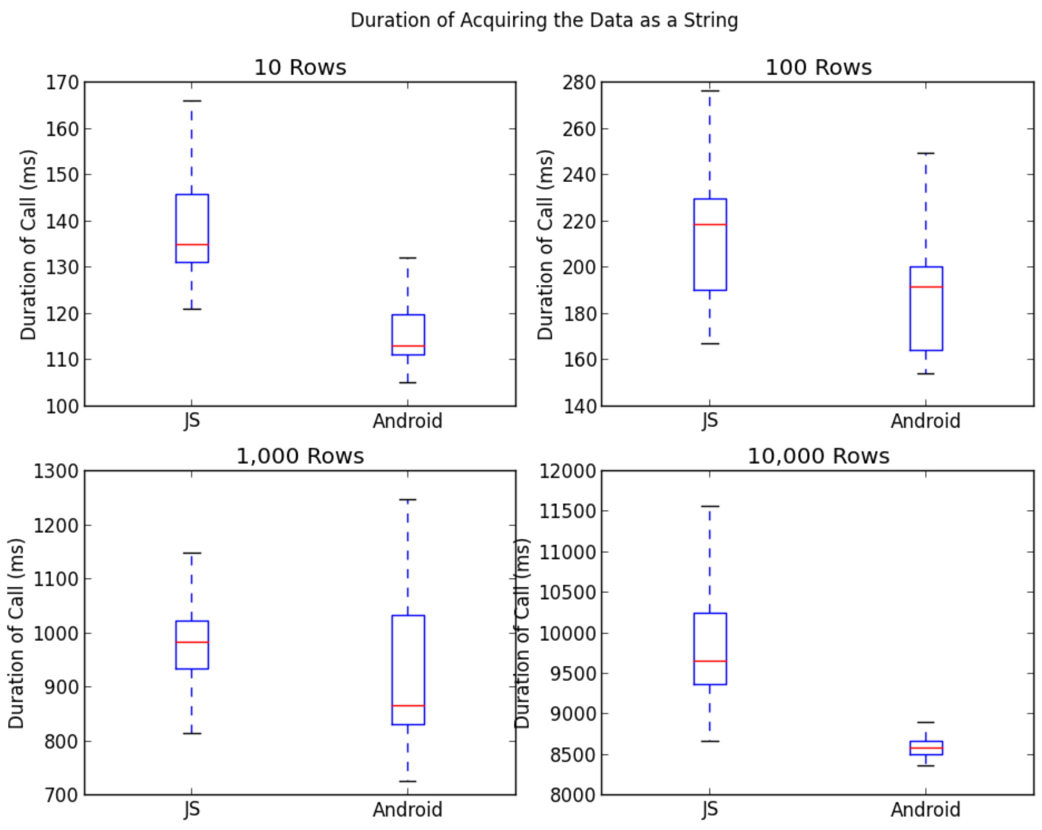


Figure 2.6: Distribution of time to acquire a string representation in JavaScript (JS) and native Android over 250 calls. Outliers have been removed.

implementation dependent. Different results are expected if the marshaling occurs in blocks of a set size, or if strings are passed using a clever semantic rather than strict serialization.

2.7.3 CPU Usage During Queries

Increased power and CPU usage was cited as a motivation for choosing C and over a browser-based experience for the Djinni framework [31]. This reasoning follows the intuition that web browsing is more computationally expensive than conventional Android app usage. The fact that the duration of object injection and of marshaling data between the Java and JavaScript

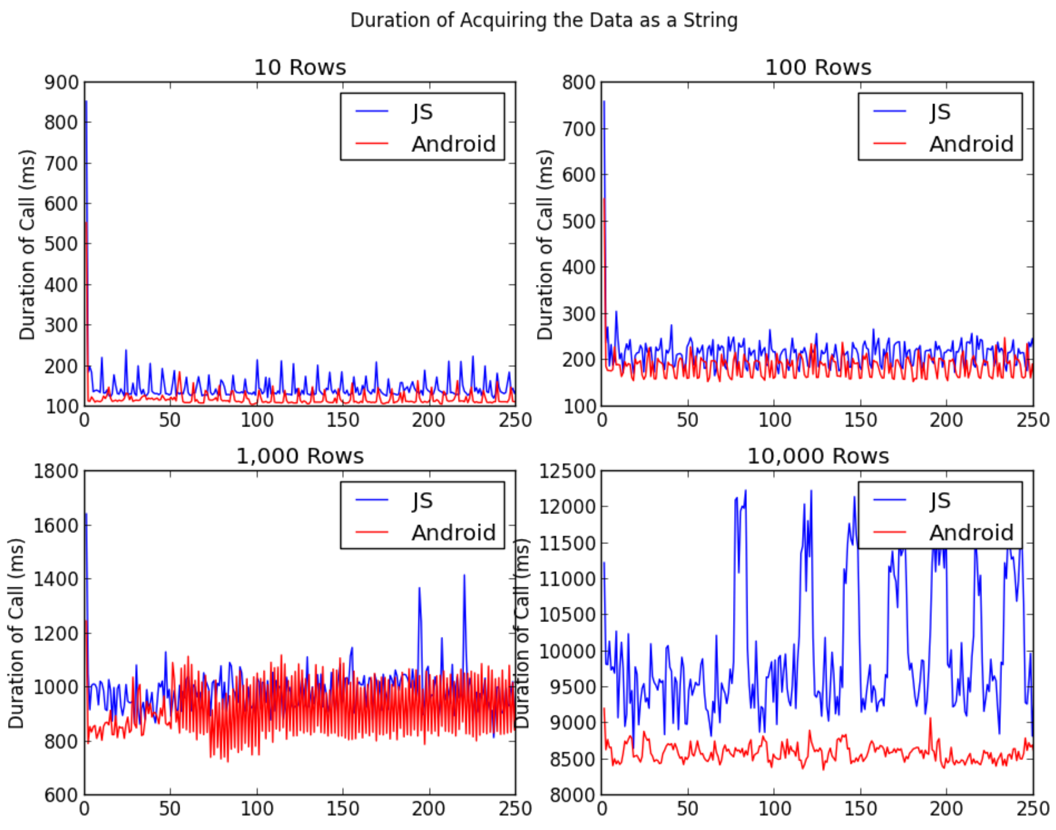


Figure 2.7: Duration of 250 calls to acquire 10, 100, 1 thousand, and 10 thousand rows in Android and JavaScript as a string.

layers do not vary by orders of magnitude suggest that perhaps the difference in CPU usage is not as significant as might be expected.

The database layer was continuously queried for a reference to an object containing 1000 rows, as well as for a string representation of that data, for approximately one hour. CPU usage was monitored via the Android adb tool, and the average percentage of available CPU cycles used by the Tables process was calculated. As summarized in Table 2.5, injecting the object into the JavaScript required an average of 32.99% of the CPU, while acquiring the reference in Java required an average of 36.58% CPU. Acquiring a String reference to the data required an average of 36.49% in native Android but only 35.46% in JavaScript.

	Android	**JavaScript**
Reference	36.58%	32.99%
String	36.49%	35.46%

Table 2.5: Mean CPU usage during an hour of repeated queries to obtain a reference to and a string representation of a 1,000 row table.

This too suggests that the performance cost of using a web approach over native Java code is not substantial, and indeed may even be competitive. If CPU usage is taken as a proxy for power consumption, Tables apps are also likely to be on par with native apps in terms of battery usage. It is important to note that no network usage and consequently radios are required by Tables apps, keeping the cost of running the web browser below what might be expected.

2.8 *Discussion*

Developers writing mobile applications repeatedly solve a common set of problems, including disconnected operation and data synchronization. Solving these problems does not relate to the value proposition of the application, and yet app designers must address these problems if they intend to create robust, data-centric mobile apps. A number of frameworks have recognized this problem and attempted to provide a set of abstractions to simplify the process of writing mobile applications.

However, these frameworks are aimed at simplifying the process for other mobile developers or teams of developers. Developers must now understand concepts like continuous queries (Yahoo's Mobius) or consider the semantics of synchronizing their compile-time applications (Izzy). Others push logic to a native C layer, requiring developers to write native code, application code, and the interface for their platform (Dropbox's Djinni). Still others aim to maintain complete support across all devices, but still expect a developer or team of developers (PhoneGap).

We have described Tables as a framework that supports the development of mobile applications by technically-inclined non-developers and lightly skilled web developers. Tables identifies a base case of mobile apps as a database viewers, providing primitives to simplify app design. It leverages web tools to achieve these goals, reducing the knowledge required to build robust mobile applications.

We have evaluated the performance of this web-based approach as compared to apps that rely on native Android code. PhoneGap is a common framework employed by the development teams of a number of major corporations, and yet this is the first comparison of web and injected code with Android code of which the authors are aware. This work simultaneously validates a common paradigm and presents a framework for technically-inclined non-developers to create fully featured mobile applications capable of functioning in resource-constrained environments.

Perhaps surprisingly, performance is comparable between Android and web-based mobile apps. Differences in call duration of as much as a second were only realized when querying tables consisting of 10 thousand rows. Motivations for making frameworks like Djinni rely on a native interface rather than a web view have been given due to the performance limitations of a web-based approach. This work suggests that such assumptions might be invalid. Few frameworks can compare to the efficiency of pure native C code. However, dismissal of the benefits of web tools, including ease of iteration and encapsulation, in the name of performance is inappropriate for a wide range of use cases. Highly performant applications are likely best written in native C or optimized Android. However, for the common use case of a data-centric mobile app as a database viewer and editor, designers can expect to not pay a heavy performance premium for using a web-based framework.

2.8.1 Native Look-and-Feel

Even in those applications that can tolerate the modest delays introduced by a web-based approach, there are limitations. The most prominent is the difference in look-and-feel between native and web-based applications. With sufficient CSS styling web components can be made to look like native UI, but the feel of the application remains different. Button presses respond differently than those in native Android. More complicated UI interactions, including gestures like swipe, have a different feel than those achieved with a native implementation. Nevertheless, this may be an appropriate set of tradeoffs for a number of users. App designers to whom platform-specific look-and-feel are crucial are likely to be the same developers or teams of developers to whom other compile-time frameworks are already available. Applications where functionality is the primary concern may be able to tolerate a less responsive swipe in the name of usability.

2.8.2 Cost of Dynamic Schemas

One of the strengths of the Tables framework is that it is completely runtime configurable. This affords app designers the power of a relational database while permitting a simpler development environment and dynamic schema creation. However, it comes at a cost.

Queries against tables of any size must first hit the database to recreate the schema information of the queried table. This means that the cost of querying tables with a small number of rows dominated by the cost of retrieving the schema, particularly for tables with many columns. This is a necessary step to properly encapsulate the persistence layer. SQLite, for example, is the native Android persistence layer but supports only five value types. Tables supports additional types in order to provide a more fully-featured storage layer.

The costs of this tradeoff can be seen in the metrics section. Queries for 10 rows were dominated by the cost of retrieve the schema information from disk. Put another way, the 30 columns in the table required reading 30 rows to recreate the definition in order to query

10 rows of data. This accounted for approximately 110ms of the 140ms for total retrieval of information from the table. This tradeoff between configurability and performance is appropriate given the motivation for Tables, although there are additional optimizations that could be performed to reduce this overhead. Some of the information surrounding a single column is serialized using a JSON representation, for instance, which is expensive. Optimization of the storage layer could be employed to lower this overhead, but a tradeoff between configurability and performance would remain.

2.8.3 Implications of Garbage Collection

One of the unexpected results of this work was that the performance of native Android code does not surpass the performance of the web-based approach in all instances. The metrics section showed that, for 10 thousand row tables, JavaScript performs slightly better, on average, than Java code in the duration of a synchronous call to the database. This is counterintuitive given that both were performing the same action, but that the JavaScript tests were performing the additional work of injecting the object. This is likely due to the difference in memory allocation made available to the application and the cost of garbage collection.

This is a difficult conclusion to predict. It shows that the when dealing with a garbage-collected language and the JVM, assumptions about performance can be complicated. From the perspective of Tables or other web-based app frameworks, this suggests that the additional layers of abstraction do not immediately introduce critical performance problems that would be avoided by using native frameworks. This can be readily seen in Figures 2.5 and 2.7, where garbage collection introduced oscillations in query time that in both Java and JavaScript. Ultimately, this suggests that web-based apps cannot be expected to perform worse than native Android applications under all circumstances.

2.8.4 Future Work

The deployments described in the Use Cases section all required participation by a team member that was familiar with building a web page. However, the Tables model affords sufficient flexibility where this should not be required for basic cases. Future work will focus on more complete integration with the app designer dashboard, which will be targeted to users with no experience building web pages. This will be accomplished through tighter integration with the provided templates and with guided creation of the declarative CSV schema.

It should also be noted that none of the factors motivating Tables dictate that it must exist as a standalone application. However, developing the tool as a standalone application has simplified the process of understanding the problem space and refining the abstractions inside Tables. This has also facilitated the performance characterization of web tools as a framework for building mobile apps, as Tables is responsible for every layer of the stack. As the interfaces between different components of the tool are finalized, it is possible that it will make sense to fold this functionality into another ecosystem or into a PhoneGap plugin. While this is not planned in the immediate future, the Tables architecture is sufficiently modular to permit this possibility.

2.8.5 Conclusion

ODK Tables was designed to simplify the creation of mobile apps, enabling technical users without software development experience to create robust mobile applications. It introduces a number of abstractions to make this possible, limiting its scope to facilitating those apps that can be modeled as database viewers. As a result, Tables apps gain high levels of customizability.

We have shown that these abstractions, along with the notion of the database row as a primitive, enable the creation of applications that are useful in a variety of resource-constrained environments. Iterating and prototyping is a lightweight process due to the use of a web tool as the UI layer. Performance and CPU usage, a proxy for power consumption, are similar in both the native and web-based approach. Therefore app designers need not expect drastic penalties when taking either approach. The Tables framework presents a set of abstractions that enables the creation of mobile apps by lightly skilled technical users, lowering the burden to app creation, and facilitating the creation of apps that can perform well in resource-constrained environments.

Chapter 3

SISKIN: LEVERAGING THE BROWSER TO SHARE WEB CONTENT IN DISCONNECTED ENVIRONMENTS

While mobile phones are widespread in the developing world, desktops and LANs still exist. This chapter describes Siskin, which is a tool designed to complement this existing infrastructure to bring web content to disconnected networks. This is especially relevant to schools, which are increasingly turning to Offline Educational Resources (OERs), employing purpose-built local hardware to serve content. These approaches can be expensive and difficult to maintain in resource-constrained settings. Siskin takes an alternative approach that leverages the ubiquity of the web browser to provide a distributed content access cache between user devices on the local network. We demonstrate that this system allows access to web pages offline by identifying the browser as a ubiquitous platform. We build and evaluate a prototype, showing that existing web protocols and infrastructure can be leveraged to create a powerful content cache over a local network.

An abbreviated version of this chapter will appear as a demo paper at the ACM Workshop on Challenged Networks in October of 2017 (CHANTS '17).

3.1 Introduction

The web has tremendous potential to enable education for users in emerging markets. With an increasing amount of free or open-use educational content becoming available online (Khan Academy, Wikipedia for Schools, etc.), schools in developing regions can bring vast quantities of human knowledge directly into the classroom. However, the web is built with the assumption of fast, free, and always-on connectivity. This presents a substantial challenge to schools in developing regions, which often have slow or intermittent Internet connectivity, such as a flaky dialup connection [1]. While modern web standards make it possible to develop sites that can be used offline (e.g., using ServiceWorkers to persist content in the browser), this approach does not scale to the vast quantity of legacy web content. The conventional approach to working around this problem involves caching static snapshots of web content, typically on dedicated hardware [24, 53]. However, this approach poses substantial logistical and cost challenges for schools in emerging markets. A typical edge cache box costs upwards of several hundred dollars, which can be prohibitive, and still requires manual maintenance and updates to software and content [37]. Schoolteachers, especially in developing countries, are not system administrators and cannot be expected to maintain esoteric hardware and software to support classroom web use.

Our key observation is that modern web browsers have the capability to support everything needed to enable automatic, distributed caching of web content that can be shared across multiple users on a LAN. While the basic idea is not new—distributed caches have been explored since the foundations of the web [86, 62, 34]—our work leverages three key insights that represent a practical, deployable solution for real-world users today. First, conventional HTTP caching is inadequate for supporting true offline access to Web content. Given that most web pages (and many resources that they depend on) are uncacheable [49], and the degree to which websites use dynamic content fetched, e.g., using AJAX, standard HTTP caches cannot guarantee that a given page will be fully cache-resident and hence usable by a user without an Internet connection. Second, separating the functionality and maintenance of the cache from the device and software used day-to-day implies that functionality will erode quickly. The caching solution should be integrated directly into the device and software (e.g., the browser) that the user interacts with, to ensure it does not introduce a single point of failure and does not suffer from neglect. And third, modern protocols (such as Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD)) enable automatic discovery and sharing of resources across a LAN segment, leveraged in products such as the Chromecast media-streaming device and others. This can enable seamless sharing across multiple users without overhead for configuration and management.

In this paper we describe *Siskin*, an approach to distributed snapshot caching of web content fully integrated into the Chrome browser. To use Siskin, the user simply needs to install a Chrome App and Extension. Siskin allows users to save snapshots of web content that they browse locally, and it automatically shares those snapshots with other users on the LAN. While browsing, Siskin automatically discovers snapshots of pages hosted by other users on the network, and it retrieves those snapshots over the local network. Siskin is built using existing Chrome App and Extension APIs to perform web content snapshotting, storage, network discovery, and peer-to-peer content transfer.

Siskin is targeted at educational settings in the developing world. Stand-alone solutions providing OERs have become increasingly popular in developing world schools. These solutions have been criticized for their difficulty to maintain and for the opacity of their content, which is typically fixed or requires significant technical expertise to update [37]. Some research has shown promise for the positive impact of these devices on education by enabling access to things like Khan Academy [1]. We make no claim on the pedagogical benefits OERs. Instead we observe that OERs are growing increasingly popular despite their cost and limitations, and we believe cheaper, more robust solutions are desirable.

We demonstrate a complete working prototype of Siskin and evaluate its performance. We show that Siskin is effective at caching snapshots of content and enabling other devices to discover and retrieve snapshots while browsing. This unlocks the potential for offline web content to be made available in classrooms with a minimum of software configuration overhead.

3.2 Related Work

Early work on web access in resource-constrained settings investigated Delay Tolerant Networking (DTN), which focused on designing network protocols to support intermittently environments that can occur in the developing world [55]. DTN approaches frequently went hand-in-hand with kiosk systems, which were custom built workstations developed to be deployed and shared in such settings [40]. Together, DTN and kiosks can be criticized as stovepipe solutions—hardware and operating systems must be custom built, and existing internet infrastructure must be modified to accommodate new protocols.

A more conventional practice on challenged networks is to install an HTTP cache. Efforts at increasing the efficiency of caching solutions have included making multiple caches cooperate [34, 62]. The C-LINK system performs cooperative caching by using a coordinating proxy to store resources using clients' local storage [54]. Wolman et al. found that collaborative caching has promise for small populations, but concluded that "the crucial problem that must be solved to improve Web performance is how to increase page cacheability" [86]. This stems from the fact that modern web pages are composed of dozens of HTTP resources (e.g. CSS stylesheets, images, and JavaScript files), and the majority of this content is uncacheable [49]. Even if those uncacheable resources are not article text, broken images or failed styling can make content hard to consume and can lower the value of content.

If caching conventions are ignored and stale resources are served, as in [54] and [16], resource-level HTTP caching remains inadequate as it assumes that devices on a network are at least partially connected, even if over a challenged backhaul. By design a cache first checks locally, and in the event of a miss it goes to the network. Since caching occurs at the network level, there is no way to inform the user of a miss. If a request misses the cache, the user might like to know that the request to the origin server will fail or will take more time than they are willing to wait. Conventional cache models do not allow this.

A number of commercial solutions have created content access hubs that enable local network sharing and host OERs. These can generally be thought of as web servers that respond to requests on a LAN. The Intel CAP, C3 Critical Links, and eGranary projects are examples [53, 24, 33]. Rachel Offline and Khan Academy Lite are software solutions built on this type of platform [69, 56]. These devices can cost on the order of hundreds of dollars, and maintenance has been shown to be a significant burden in resource-constrained settings [37]. Our work exists as a complement to these efforts, demonstrating that similar functionality can be provided without stand-alone hardware and without introducing a single point of failure.

Our work is most similar to [16], who implement aggressive HTTP caching as a Firefox extension. That project does not coordinate between machines, preventing users from benefiting from peer caches on the network. It is also aimed at accelerating browsing behavior, e.g. through aggressively prefetching links on a page from the internet, rather than on distributing content. For these reasons their system is not well-suited as a platform for OERs.

3.3 System Design

Siskin provides a seamless distributed cache of web page snapshots available to any device on the same LAN segment; this will typically be a single classroom or a group of classrooms. Every peer in the Siskin network hosts web page snapshots, which can be discovered and fetched by other peers on the network, avoiding the need for fetching content from a slow or intermittent Internet connection. Siskin achieves this using a combination of local page snapshotting and caching; peer discovery using mDNS; peer-to-peer snapshot fetching using Web Real-Time Communication (WebRTC); and a Chrome App to provide the UI. In this section we provide details on Siskin's architecture and implementation.

Our prototype is implemented as a Chrome App and Extension combination that communicate via the App and Extension APIs. Both components are needed due to the security model of Apps and Extensions in Chrome. Extensions are able to interact with a user's web browsing experience, e.g. to save a page as MHTML or to inspect navigation requests. Apps are able to access the system directly, e.g. to write files to disk and access system sockets.

3.3.1 Snapshotting Content

The fundamental unit in Siskin is the rendered page. This creates a one to one mapping between a top-level URL and a resource resident in the cache. This approach effectively snapshots the rendered web page that results from a visit to a URL. Unfortunately there is not yet a standard for distributing web pages that perfectly recreates a connected experience.

The most widespread support is for MHTML, which takes a snapshot of the DOM, inlining external resources like images. This has the benefit of being a single hermetic file, making distribution simple, and of being well-supported by browsers. When a saved page is viewed, it is fetched from the peer, saved to disk, and displayed in a browser tab.

In our prototype, MHTML snapshots are saved manually by users. When visiting a page, clicking the Siskin Extension icon in the Chrome toolbar saves the page as MHTML and adds it to the cache. This uses the `pagecapture` API to snapshot the page as an MHTML blob. The blob is then passed to the App via the `runtime.sendMessage` API, where the App saves the page to disk using the `fileSystem` API. For security reasons, we elected to keep snapshotting a manual process. This is discussed in Section 3.3.5.

Alternative content ingestion methods are possible. A set of pages could be crawled and saved as MHTML, snapshotting them, and then moved into the Siskin directory on a host machine. This versatility allows existing content from OERs, including Rachel Offline or eGranary, to be shared using Siskin.

Although widely supported, a notable shortcoming of MHTML is the fact that it does not support JavaScript execution. For this reason responsive sites and web apps are not well-suited to MHTML. An alternative approach might be to ignore cache-control headers and simply cache all resources, as in [16] and [54]. We find the single file, hermetic nature of MHTML to be a preferable distribution mechanism. Resource-level caches are best suited to configurations where a cache sits between a machine and the origin server, allowing individual requests to be handled in flight. Saving pages as an MHTML resource simplifies distribution by allowing alternative configurations, including look-aside caching behavior where a user is informed before navigation that a local load will succeed. Employing MHTML allows content to be added to the cache as logical units and distributed with associated provenance. It is explicit that MHTML is only a snapshot, not a stale page served from a cache.

Siskin does not try to support web app behavior that requires complex interactions with a server. Email applications, for example, are not handled. Siskin aims only at operations where a page can be displayed without needing to interact with a server after the time the page is saved.

3.3.2 Peer Discovery

A discovery component is necessary to find peers running Siskin on the local network. We accomplish this by employing mDNS and DNS-SD. These zero-configuration protocols are a standard solution to the problem of network service discovery; they are employed by many services, including Chromecast media streaming devices [19, 18]. mDNS uses multicast UDP to issue DNS queries to the local network, while DNS-SD specifies how to use DNS records as a hierarchical database for service discovery. Using both together, clients can discover peers running a service and resolve an IP address and port combination to connect to the service. Chrome Apps provide an API to issue and respond to UDP requests, making an implementation of mDNS and DNS-SD straight forward.

Alternatives to mDNS and DNS-SD exist. Simple Service Discovery Protocol (SSDP) is another solution that relies on multicast addressing to query and advertise via the local network. In some use cases, seamless discovery is not required and the peer to peer process can be bootstrapped by directly sharing connection information. A teacher might write the IP address of their host machine on the board, for example, or a WebRTC offer could be shared directly between users. Distributed systems such as [75] communicate with peers without keeping IP and port information for each individual device. These have desirable properties for large systems, namely that contact information is only needed for $O(\log N)$ peers to communicate with content that could be stored on any of the N peers. This is not an ideal solution for Siskin, which seeks to provide full peer enumeration as a desirable property.

3.3.3 Content Discovery

Local content discovery takes one of two forms. In the first, a list of peers on the network is presented via the App UI. Upon selecting a peer, the list of pages saved by that peer is presented to the user much like listing the contents of a directory. In the second, regular browsing behavior is augmented to inform users of locally available content. This process is referred to as "cache coalescence". To accomplish this, peers disseminate Bloom filters, as in [34], representing the URLs they have cached locally. The App component of Siskin maintains this information, allowing instances to locally determine if a given link is available from a peer. Links on a page are annotated to show that they are available.

By providing both modes of content discovery, Siskin is able to support sparse cache occupancy, where a user might save only a single page, as well as dense cache occupancy, where a whole domain may have been snapshotted. If a particular user has saved part of Wikipedia, for instance, users can first discover an entry point into that content via the summary listing. After navigating to the page, link annotation will allow the user to become aware of local content by augmenting regular browsing behavior.

Siskin annotates locally available URLs by communicating between the Extension and App modules. The Extension includes a ContentScript—a standard Extension component—that runs on on every page the user browses. This ContentScript queries the page for anchor tags that include the `href` property. These URLs are passed to the App using the `chrome.runtime.sendMessage` API. The App holds the cache state of peers in memory as an array of Bloom filters. Each URL is checked in memory to determine if it is likely available on the network (the use of Bloom filters make this a probabilistic operation). Matches are passed back to the ContentScript, where the anchor tags are annotated with a small cloud icon to indicate their availability on the local network. More efficient alternatives, including distributed indexing as in [75], could be layered on top of Siskin. However, as discussed Section 3.4, that is not necessary for the contexts we are considering.

3.3.4 Data Transfer Between Peers

Data must be transferred between peers to perform content discovery and to view the snapshots. Our prototype employs WebRTC as the transport mechanism. WebRTC is a protocol suite designed to enable peer to peer communication between browsers. We chose WebRTC because it is built into the browser, and because all communication over WebRTC connections is encrypted.

Negotiating a WebRTC connections requires a signaling step, which is an exchange of text blobs referred to as “offers”. In our prototype implementation, WebRTC offers are exchanged via an HTTP endpoint using an open source HTTP server [85] we bundle with the App. The initiating peer issues an HTTP PUT request, the body of which contains the offer. The peer receiving the request extracts the offer, generates a return offer, and includes this offer in the HTTP response. With offers exchanged, a WebRTC connection can be established. We layer a messaging protocol on top of the connection’s data channels, and Siskin instances use these messages to communicate.

3.3.5 Security and Privacy

We identify three main areas relevant to security and privacy in Siskin: 1. confidentiality of shared content; 2. secure communication between peers; and 3. integrity of cached content.

The first area relates to the confidentiality of content. Our approach of snapshotting pages makes this problem non-trivial. If a user elects to snapshot their email client or social media wall, for instance, they run the risk of leaking potentially private information to others on the network. We elected to make snapshotting pages require a manual action from the user. In our prototype, content is guaranteed not to be shared until hitting the extension icon and electing to save content, allowing users explicit control over what content is available. Additional defenses could include blacklisting social media sites and implementing access control, allowing users to snapshot content for only their own use or to only share it with particular users.

The second area relates to secure communication. Siskin should support both encrypted and authenticated communication. The WebRTC transport mechanism ensures that communication is encrypted, but it is not authenticated. In connected contexts, authentication occurs during the signaling step by communicating through an identity provider or via an HTTPS connection. This is complicated by the offline settings Siskin targets. However, we could perform authentication by exploring a protocol akin to Secure Simple Pairing (SSP), which is used to pair Bluetooth devices. It provides a way to both exchange an encryption key and authenticate that the exchange was not subject to a man in the middle attack. SSP includes a numeric comparison mode by which two users compare several numeric digits, checking for equality. This exploits physical proximity of two devices and is appropriate for Siskin, where users are expected to be on the same LAN. SSP has been shown to be secure [60]. After SSP is complete, private keys can be shared and future secure communication can occur using RSA, provided that keys remain confidential.

The final area, integrity of content, is not provided by our prototype. Snapshotting and sharing are performed by untrusted peers, meaning that integrity guarantees stemming from the use of HTTPS are lost. Snapshots could be tampered with, or sophisticated peers could falsely claim that a fabricated snapshot originated from a specific domain. Integrity guarantees could be maintained by adding a level of trust to peers and cryptographically signing snapshots. An alternative could be to use a third party service to generate and sign snapshots. These approaches would require additional infrastructure and are not things we currently support.

3.4 Evaluation

In this section we perform a technical evaluation of our Siskin prototype. We evaluate the capabilities of the technology in a laboratory setting, showing that the architecture of our system meets our design requirements and could be employed as an alternative to standalone OER solutions. All our experiments were conducted on Acer 14 Chromebooks on the network in the University's Computer Science department. The Acer 14 is a mid-level Chromebook available for approximately $300 USD. The models used in our evaluations had a 1.6 GHz Intel Celeron N3160 Quad-Core Processor, 4 GB of RAM, and were running Chrome OS 58.

Metrics that depend on the number of pages in a cache are calculated for 1,000 pages. This number was chosen because many of the most highly rated Rachel Offline modules, which serve as OERs, contain on the order of 1,000 pages [69]. Additionally, 1,000 pages is a realistic number of pages for a user to add manually without an automatic content ingestion mechanism to add pages to Siskin.

3.4.1 Data Transfer Speeds

Data is exchanged between Siskin peers both to communicate operational data like cache state as well as the saved pages themselves. Siskin relies on browser JavaScript APIs for storage and network communication. The key question is whether browser-based peer-to-peer data transfer is fast enough to serve cached pages between peers. To measure transfer speeds, we installed Siskin on two Chromebooks and connected them to the department's 802.11a/g/n WiFi network.

We generated files of 1 kB, 10 kB, 100 kB, 1 MB, and 10 MB, and measured the time to transfer each file 100 times between two peers. This was conducted using two transport mechanisms: WebRTC and WebRTC including connection establishment. In WebRTC connection establishment is referred to as signaling, and is required for first time or infrequent requests to a peer. Connection establishment requires one additional round trip between peers. Mean transfer times over 100 runs are shown in Figure 3.1. The results show that each of the transfer methods achieves performance roughly in line with the capacity of the underlying WiFi network.

3.4.2 Query Latency

Next, we measure the latency for querying the distributed cache for a set of URLs. When a page is rendered, Siskin scans the page for all links contained in anchor tags in the page, and queries the cache for the existence of each URL, so it can annotate the page with the cache residency status of each link. One of the key advantages of Siskin over conventional HTTP caching is greater transparency of available content. URLs entered into the address bar as well as URLs from anchor tags on loaded pages are queried against the body of locally available content.

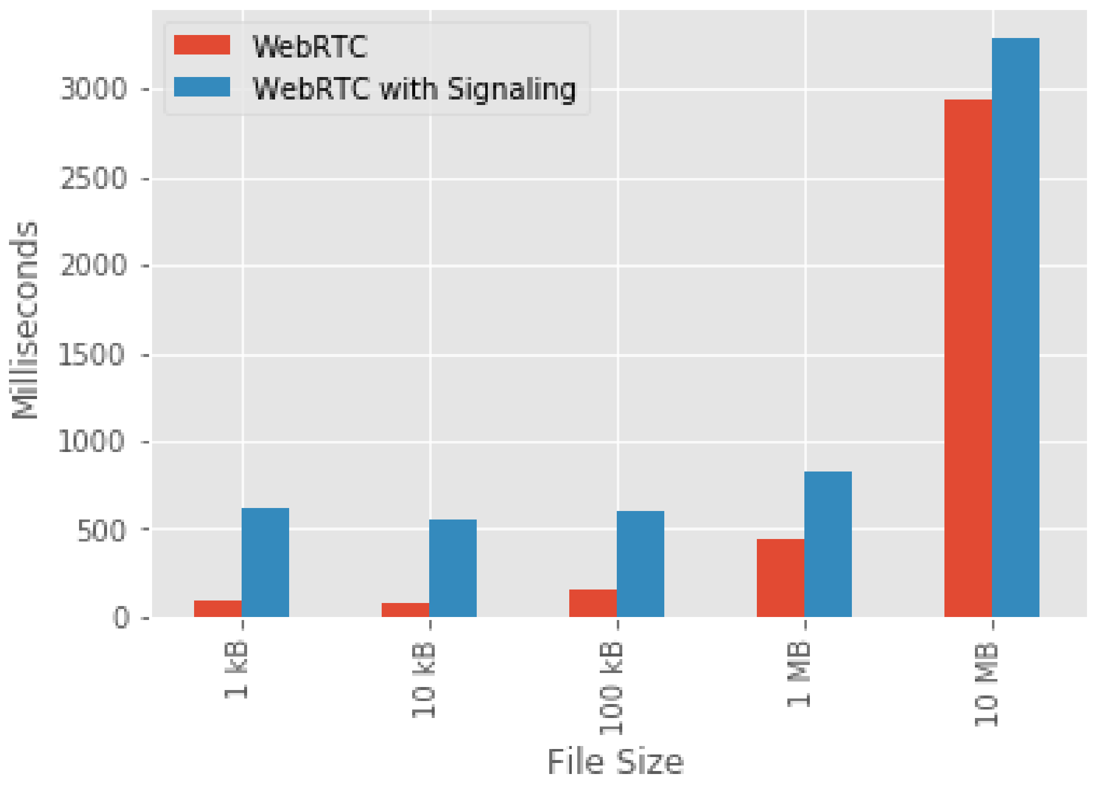


Figure 3.1: Mean transfer times for various file sizes between two peers.

To measure query latency, we constructed four HTML pages with 1, 10, 100, and 1,000 outbound links. We also generated a synthetic cache directory containing 10 peers, each of which contained 1,000 pages. Each of the HTML pages was loaded 100 times. The cache directory was already present in memory, as if peer state had already been communicated. The query latency is the time to look up all of the outbound links on the page in the cache directory.

The mean results over 100 runs are shown in Figure 3.2. Querying up to 100 links takes on the order of 100 ms, while querying for 1,000 links takes on the order of 600 ms. As shown in the Figure, the time it takes to find the links and query the data structures grows with the number of links, as can be expected. Communication between Chrome Apps and Extensions occurs via a messaging and callback system. Waiting for these messages to be delivered and callbacks to be invoked by the Chrome machinery constitutes the largest component of the query latency.

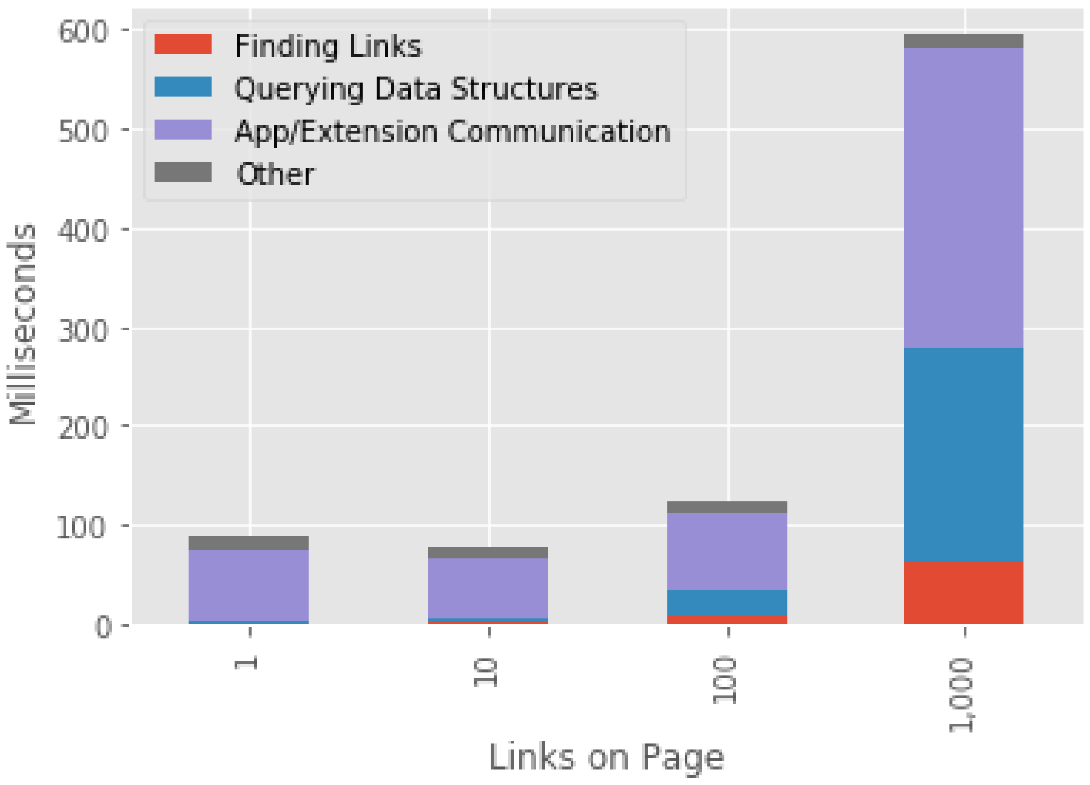


Figure 3.2: Mean times to query for URLs available on the network of 30 peers, each with 1,000 pages.

3.4.3 Overhead for Cache Coalescence

Finally, we evaluate the overhead of distributing cache directory information between peers in the network. As the number of peers and the size of the cache grows, the question is how much local network bandwidth is required to distribute this information. Our prototype of Siskin uses a fairly simplistic cache coalescence strategy—each peer exchanges a Bloom filter of locally-cached URLs with every other peer on a periodic basis (every 60 seconds). We perform this via unicast, where each peer sends a Bloom filter to every other peer. However, more efficient schemes, for example, leveraging multicast, are possible as well.

We analyze a network of between 2 and 50 peers, each of which is caching 1,000 pages locally. (In reality, we would expect most peers to store very few pages, perhaps no more than 100, whereas one or two “super peers” might cache a large archive of pages.) Each peer encodes the list of cached URLs into a Bloom filter with a target false positive rate of 0.001, which requires 1,798 bytes. We consider an 802.11b network, as might be expected at a rural school, with an aggregate TCP throughput of 5.9 mbps. Maximum broadcast throughput is considered to be 0.5 mbps—one half of the lowest supported 802.11b rate of 1 mbps.

Under the fairly simplistic unicast system implemented in our prototype, fully distributing cache state requires each peer to communicate their state to every other peer, resulting in 4.4 MB that must move across the network. On the 802.11b network described above, this would require 6.0 seconds if all 50 peers joined the network at the same time. Figure 3.3 shows the bandwidth requirements of our unicast strategy for different refresh rates. This assumes that an initial distribution has been completed and the Siskin peers are periodically informing peers of their content by completely redistributing their Bloom filters.

With a 60 second refresh rate, the bandwidth impact is minimal. However, the unicast strategy is naive in that it requires an $O(N^2)$ operation in the number of peers, as each peer sends their directory information to every other peer. This can be improved by employing a broadcast mechanism to transmit the Bloom filters. Under this scenario each transmits a Bloom filter a single time, for a total of 0.09 MB. Given a broadcast throughput of 0.5 mbps, full distribution would take 1.4 seconds per distribution cycle. Figure 3.4 shows the bandwidth requirements of a broadcast coalescence strategy for different refresh rates. This assumes that an initial distribution has been completed and the Siskin peers are periodically informing peers of their content. With a 60 second refresh rate, the bandwidth impact is minimal.

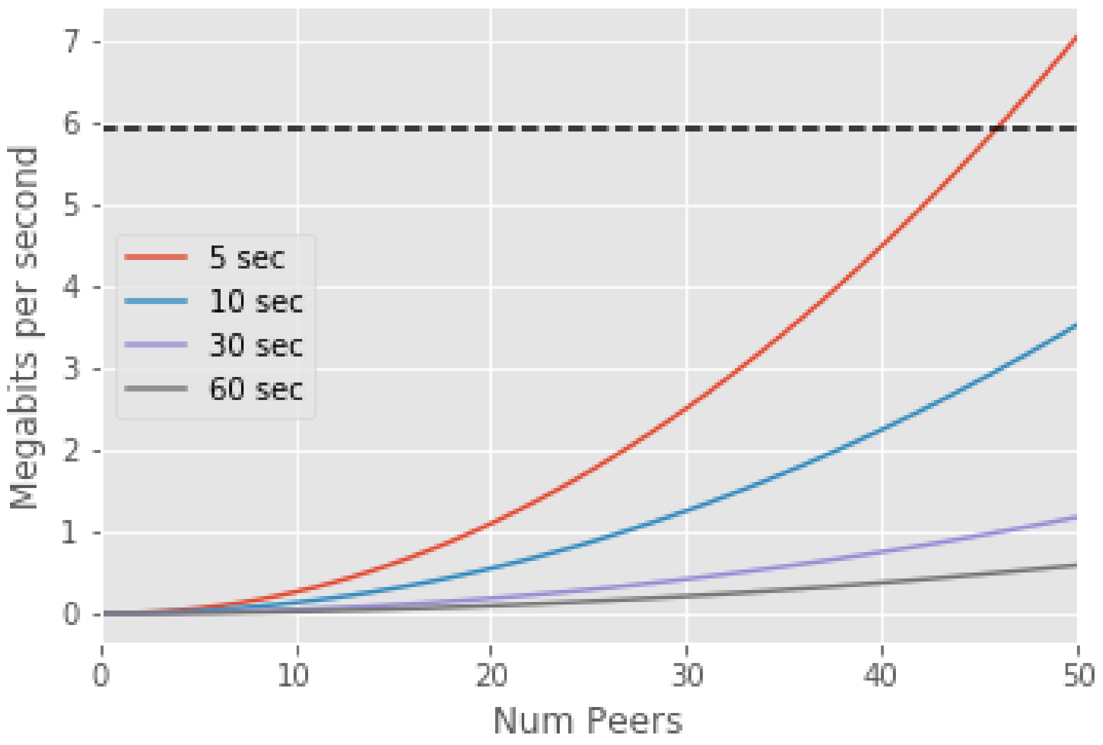


Figure 3.3: Bandwidth consumed by fully redistributing cache directory information via a unicast mechanism. Bandwidth is estimated at different refresh rates as additional peers join the network, each hosting 1,000 pages. The dashed line shows a theoretical maximum throughput of 5.9 mbps.

3.5 Discussion

Siskin shows OERs do not require expensive, purpose-built hardware. It demonstrates that the web browser has become a sufficiently ubiquitous and capable platform that it can serve as a powerful offline content caching and distribution system. However, it is not a perfect solution. In this section we outline limitations of the system, how those limitations could be mitigated, and what remains as future work.

The most immediate limitation is the capabilities of of MHTML as a distribution format.

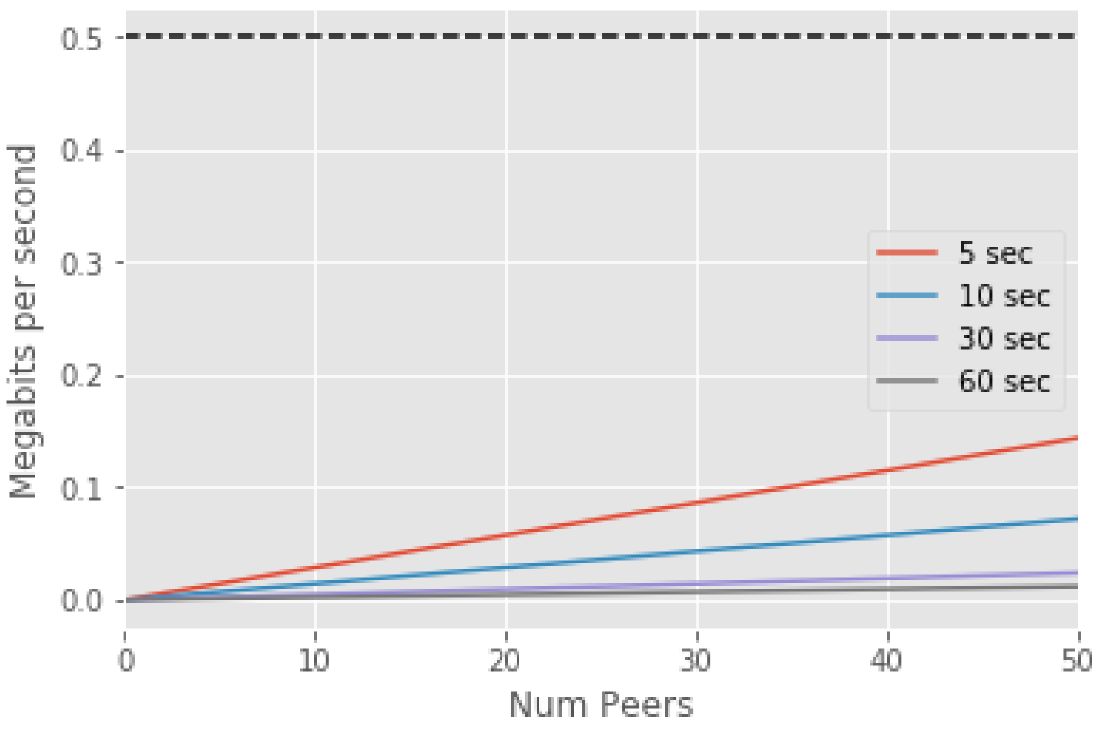


Figure 3.4: Bandwidth consumed by fully redistributing cache directory information via a broadcast mechanism. Bandwidth is estimated at different refresh rates as additional peers join the network, each hosting 1,000 pages. The dashed line shows a theoretical maximum broadcast throughput of 0.5 mbps.

MHTML does not run scripts, meaning sites that depend on JavaScript after an initial render will not work as expected. This does not impact all sites, and MHTML continues to be used in industry settings, but it is nevertheless a limitation of the format.

A more pressing problem is the fact that MHTML does not support video. Sites like Khan Academy are popular as offline educational resources [1]. Khan Academy lessons, as well as video services like YouTube, are not well served by MHTML. The DOM is saved but the media player is replaced by an empty HTML element. This could be mitigated by storing video files separately from MHTML and alerting the user to the associated files.

This still would not be perfect, however, as the experience would differ from standard web browsing. A long term solution would be to develop a file format that better handles modern web content.

The current usage model of Siskin is based on URLs. Snapshotted URLs that are available can be listed, and links can be redirected to local copies. A useful feature would be the ability to perform keyword searches. Wikipedia becomes much more useful, for example, with the ability to search. Without search, users would have to rely on the directory listing feature of Siskin to find an entry point into cached content. Alternatively, a manual index page, like a site map, could be employed to explore content. Searching in an offline setting is complicated by the fact that content can join or leave the network with hosting machines. This suggests that machines should host their own search indices. Creating these indices in a way that serves real-world workloads while respecting the performance and storage capabilities of local hardware is left for future work.

The primary motivation behind Siskin is the idea that the ubiquity of web browsing technology can create distributed content distribution mechanisms. Our prototype, however, depends on Chrome App and Extension APIs. This means that it does not work on mobile devices, which do not support these APIs. An ideal Siskin implementation would run on any browser, not just laptops and desktops. Siskin is an example of how the browser can be used to replace stand-alone solutions. Our prototype demonstrates that this is feasible. We do not claim that our prototype is a complete solution. Instead we claim this demonstrates how the browser as a ubiquitous technology can expand the reach of web content even to those that are only intermittently connected.

3.6 Conclusion

We have presented Siskin, a system that supports OERs at schools in the developing world without requiring additional hardware or purpose-built devices. It achieves this by facilitating the distribution of web content on intermittently connected networks. Siskin leverages the ubiquity and capabilities of the web browser to distribute content between peers and integrate with the web browsing experience. Siskin represents the ability to replace or complement stand-alone solutions with existing, ubiquitous infrastructure—the browser—to give disconnected users access to the more than four billion pages that exist on the web.

Chapter 4
THE SECURITY OF DATA COLLECTION TECHNOLOGIES

This chapter moves the focus from the technology itself to the people using the technology. It examines organizations using Open Data Kit (ODK) to understand their approaches to security in their deployments. This work was conducted as a joint work with Camille Cobb and I present it with her permission, as well as with some small modifications. It originally appeared in 2016 at the International Conference on Information and Communication Technologies and Development (ICTD '16) [22].

4.1 Introduction

Although there have been some efforts to study and address computer security and privacy risks with technologies in an Information and Communication Technologies for Development (ICTD) environment, both on a case-by-case basis for specific technologies and from an academic perspective, e.g., [2, 23, 71], the space of "computer security meets ICTD" is still in its infancy. This is a gap in the field. As discussed in Chapter 1, the technology and users that are implicated in ICTD can have different expectations. Without examining these differences, security concerns could be mismatched, creating a situation where tools and security models are not well-suited to their context. We contribute to this space through insights into how to evaluate and address computer security risks in ICTD environments.

To provide a foundation for our insights, we focus on a particular class of technologies—data collection toolkits—and, in particular, a specific, widely-used instance of such a technology: Open Data Kit (ODK). Data is crucial for many NGOs and researchers to monitor and evaluate deployments or interventions and report to donors on activities. For example, organizations might collect patient information during clinic visits, assess the prevalence of pests in rural farmland, or document infrastructure in need of repair. ODK allows digital forms to be created without deep technical expertise, and has been used as a platform by numerous organizations. By studying computer security risks with ODK, we are able to extract lessons for both ODK and other data collection deployments, as well as infer lessons for other new ICTD technologies.

This work is a collaboration between an ICTD research group and a computer security and privacy research group and leverages methodologies from both communities. For example, our threat model for data collection technologies (Section 4.4) is the result of a large *threat modeling* process (used in computer security) that involved many members from both groups. We augment that with surveys and semi-structured interviews, and leverage our past experiences (within our ICTD research group) in conducting deep investigations with key stakeholders (Sections 4.6 and 4.7). Our threat model provides an analytic overview of the potential issues for data collection technologies, and the surveys and interviews provide a context within which to appropriately interpret and evaluate the risks of threats that we identified.

Contributions Our contributions are three-fold. First:

- *Threat Model.* We develop a threat model for ODK and other data collection systems.

Our threat model provides a broad, encompassing view of the *possible* threats to an ODK-like system, including possible adversaries and adversarial methods. However, computer security is not a binary property and just because a computer security attack *might* be possible does *not* mean that it is likely to happen in practice. Hence, a more nuanced approach to computer security is to not only identify the possible threats, but to understand the broader context for a deployed system. Providing an informed, broader contextual analysis is our second key contribution:

- *Survey and Semi-structured Interviews.* We report on a survey and semi-structured interviews with ODK deployment architects.

We use the results of the survey and interviews to extract insights into how ODK deployment architects think about security. "Think about security" is intentionally broad; we consider, for example, not only how deployment architects perceive threats, but what defensive mechanisms they have deployed and why, what incidents they have encountered and how they responded, and so on. Finally:

- *Broader Synthesis and Recommendations.* We consider overarching implications and recommendations.

Among the key takeaways: as in the developed world, computer security of data collection platforms in the developing world is about risk management. Though our survey and interviews surfaced real threats and security concerns—particularly about data loss and erroneous or fake data—many of the threats we consider abstractly seem to have not yet manifested in practice for many of the deployments we studied. Hence, the current level of security seems arguably appropriate in today's environment, particularly given the practical tradeoffs faced in balancing security with other deployment goals. However, ICTD systems (and their data) may persist for many years, and the risks may change over time, making it important for organizations to proactively consider and revise their threat models.

4.2 Background and Related Work

On computer security and privacy Since this work is a joint effort between an ICTD group and a computer security research group, we take a moment to establish common terminology. A key point of terminology is that, for this paper, when we say "security" we refer to computer security (also commonly referred to as cyber security); for others, the term "security" might invoke other notions. For example, in some ICTD contexts, "security" evokes thoughts of "food security", meaning enough food to eat. Because of the possibility for different interpretations, we were cautious with terminology both during our interviews and in our later evaluation of the interview results. Second, we note that the computer security community often use the terms "security and privacy" in unison, because security vulnerabilities can lead to privacy compromises; other communities have more nuanced usages of the term privacy. In this paper, we use computer security community's interpretation of the term privacy.

Related work Ben-David et al. [2] focus on end-users in the developing world and argue in a position paper that five factors influence how security threats differ from those in the developed world. Some of the security work in ICTD has looked at specific threats brought on by weak security infrastructure and hygiene, such as sharing content by USB on unpatched systems [23, 3], which is a concern in the domain we study. Security researchers have studied apps used for mobile banking in the developing world, and found them rife with security vulnerabilities [71]. Mancini et al. [63] explore technical guarantees that might be afforded to digital data collection with a new system of APIs designed to be secure. Gejibo et al. [38] describe how low budget phones could securely store data during mobile data collection. Gejibo et al. [39] also describe how a cloud-based server could store data securely. Hussain [50] describes the sensitive types of data that organizations collect and references the legal code of several countries to argue that it is worth securing.

Security researchers have studied the security practices of specific user groups (e.g., [65, 25, 12]), but to our knowledge, the group we consider here (users collecting data in an ICTD context) has not be studied from a security perspective. Le Blond et al. [59] report on the characteristics of targeted cyber attacks on an NGO.

4.3 Open Data Kit: Background

We focus on Open Data Kit (ODK) as a prototypical data collection application used in developing regions. Chapter 2 focused on ODK Tables, which is a component of a new tool suite under the ODK Umbrella. Broadly speaking, the tools are sufficiently similar that security lessons apply to both suite of tools. This section provides additional background information on the original tool suite, as most respondents are more familiar with the original tools.

ODK was created in 2010 by researchers with the goal of providing a general purpose tool to facilitate data collection in ICTD contexts [45]. It has been widely adopted, used in at least 125 countries and installed on hundreds of thousands of devices [80]. ODK allows digital forms to be created without deep technical expertise. It supports traditional text and multiple choice questions and leverages sensors on the devices to capture rich data types, including GPS locations and photographs. Factors contributing to its success include the fact that it is adaptable to a variety of settings and data domains and is free and open source [42, 45, 10].

Basic ODK setup involves: (1) creating an XML form for each questionnaire using a graphic user interface, (2) downloading the ODK mobile application to an Android device, (3) setting up an ODK (local or cloud) server using a simple installer, and (4) downloading the XML forms to the mobile devices. Then, the deployment involves: (1) filling out these forms on mobile devices, from which data being is to the SD card, (2) syncing to the server when there is a data connection, and (3) accessing/analyzing data on the server.

Stakeholders The typical roles, or stakeholders, in an ODK deployment are identical to those discussed in Chapter 1. They include: (1) donors that fund the work; (2) deployment architects that create the forms, administer deployments, and make technical decisions, possibly based on input from an ethics board or organizational policy; (3) enumerators that complete surveys on mobile devices; and (4) beneficiaries that provide data to enumerators. These are representative of the roles someone might have in a deployment, though not all roles are necessarily represented in every deployment, one person might take on more than one role, and each role may be filled by more than one person. Within a deployment, individuals in each of these roles may or may not be trusted or trustworthy and will likely have a wide range of technical skills.

4.4 ODK and Computer Security

Having summarized key properties of ODK's design, we now proceed with a deeper analysis of its computer security properties. We consider this *security analysis*, with roots in the *threat* *modeling* process common in the field of computer security, as a contribution to the ICTD community for two reasons. First, it surfaces key computer security threats and opportunities that we believe ODK deployment architects should consider. Second, it expands on other threat modeling work in ICTD (e.g., [2, 23, 71]) and provides a data point for threats that might arise for ICTD applications.

Our security analysis in this section is done in a theoretical, abstract sense: we consider potential computer security threats that *may* arise in an ODK deployment. In practice, no system is completely secure, and computer security consists of a series of tradeoffs and risk management. It may be the case that some threats are worth defending against (because the cost or risk of compromise is high) whereas others are not. However, it is important to identify a superset of possible threats in order to enable informed decisions about these tradeoffs. We provide this superset in this section; our surveys and interviews in Sections 4.6 and 4.7 then provide insights into the relative importance and likelihood of these risks and threats, as perceived by current ODK users.

4.4.1 Potential Threats to an ODK Deployment

Threat modeling is a process commonly used in the computer security community by which one identifies potential adversaries and their motivations, as well as potential threats and vulnerabilities. Our threat modeling for ODK results from a collaboration between experts in the computer security and ICTD communities. Our approach involved systematic brainstorming discussions and an empirical analysis of possible vulnerabilities in an archetypal ODK application (written specifically for this purpose). We present the results of our threat modeling exercise, stressing again that the issues we raise here are a superset of the possible issues that ODK deployment architects might face in practice.

Security and privacy goals We begin by identifying possible security and privacy goals that stakeholders in an ODK deployment may have. Computer security literature often refers to the "CIA" goals for computer security: confidentiality, integrity, and availability.

1. *Confidentiality.* An adversary should not be able to learn private information about individuals or sets of individuals whose data is collected as part of the ODK system. We can consider varying levels of confidentiality, e.g., it might be OK for some adversaries to learn aggregate information (such as the total number of patients) but not individual information (e.g., the records for a specific person). Confidentiality might also apply to the enumerators (e.g., the healthcare workers)—an enumerator may not want their location or time of data collection disclosed to some adversaries.
2. *Integrity.* An adversary should not be able to cause false information to be collected or stored as part of the ODK application. These adversaries might include enumerators trying to avoid doing their work, beneficiaries lying to enumerators, application developers surreptitiously modifying data after collection, and so on. Guaranteeing that false information is never collected may be difficult or impossible in general. An alternate version of this goal may be: it should be possible for the managers of the deployment or the data analysts to detect and/or mitigate discrepancies due to false data collection.
3. *Availability.* Data, and the ability to collect data, should remain available even if a device is disconnected from the Internet for an extended period of time, or if the device is lost or stolen. Remote access to servers should be robust to denial-of-service attacks.

Potential adversaries We next identify potential adversaries and adversarial goals to an ODK deployment. Potential adversaries include any stakeholders of the deployment itself, as well as external actors. For example:

- *Enumerators* may provide fake data in an attempt to simplify their own jobs, or may violate the confidentiality of data provided by a beneficiary by disclosing private information (e.g., HIV status) to someone not intended to learn the information (e.g., a spouse).
- A given deployment may involve multiple *partners* who are involved with different parts of the deployment and are intended to have access to different forms and/or data. A

malicious partner might violate this intention.

- *Governments or other powerful organizations* may target sensitive ODK deployments (e.g., those collecting information about government-related opinions) in order to learn the identities of or information about stakeholders involved in the deployment.
- *Other adversaries* not targeting the deployment specifically may nevertheless cause problems. For example, external hackers may attack the deployment servers and corrupt or steal the data or take the server offline, or thieves may steal mobile devices involved in the deployment for the hardware, resulting in the loss of data.

Potential threats Finally, we consider concrete threats that may result from the above adversaries. For example:

- Unauthorized access to forms or data on the device, or to the remote server, to access, modify, or delete data
- Entering fake data into a form (maliciously or accidentally)
- Coercing or bribing enumerators or other deployment stakeholders to reveal sensitive information about the deployment or beneficiaries
- Physical theft of a mobile data collection device
- Legal access to data, e.g., through a subpoena
- Inability to use the data collection application
- Fake ODK applications on software marketplaces
- Improper disposal of devices used in data collection
- Other malicious applications installed on devices
- Information leaking to other applications on the device
- Denial-of-service attacks preventing data from being uploaded to the server

Some of these threats are not solely hypothetical. For example, by default ODK data is stored in plaintext on the SD card; this data can easily be extracted from the device and is world-readable in some versions of Android.

4.4.2 Possible Defenses

We now turn to possible defenses. Some of these defenses are already available to ODK deployment architects, some embody standard best practices but have not been incorporated into ODK yet, and others employ either new ideas or ideas from the computer security literature.

Available defenses and best practices Existing security measures supported by ODK or otherwise available on Android include:

- Encryption of saved ODK data
- Encryption of the Android device
- Encrypted connection to the server (i.e., TLS/SSL)
- Access control for server access
- Android apps to lock down phone capabilities
- Checks to prevent or detect fake data entry
- Locking the phone screen
- Keep software up to date

Additional defenses We considered a set of possible additional defenses. We report here on those that we later discussed with participants because we thought they had the most potential to be relevant to a variety of deployments. These additional defenses include panic passwords, geographic restrictions, and dongles to replace or augment passwords. Panic passwords are passwords that can unlock a device but simultaneously trigger an alert, erase data, or present synthetic data [21]; panic passwords are particularly useful when users might be coerced into unlocking a device against their will. Geographic restrictions refers to the notion of limiting certain functionality, e.g., data collection or access, to specific geographic regions. Using authentication dongles instead of passwords can lead to increased usability, assuming that the user has a dongle in their possession; using dongles in addition to passwords can provide greater security than passwords alone.

4.5 Methodology

In Section 4.4 we surfaced a spectrum of threats and security considerations potentially applicable to ODK-based deployments. However, as noted earlier, not all threats are equally likely nor have the same impact. Some threats may never manifest because they are either too costly for an adversary or the rewards to the adversaries are too little. Given this nuanced perspective, a key question thus arises: which threats should a deployment seek to mitigate?

This question leads to the second major component of this work: to provide an informed understanding of how ODK users currently perceive the computer security landscape. We designed a two part study—a survey (Section 4.5.1) followed by in-depth interviews (Section 4.5.2)—to better understand these issues. We received approval from our institution's human subjects review board to conduct our surveys and interviews. We have also disclosed the results of this work to the core ODK development team, a common practice in security to ensure that developers can react to system vulnerabilities that may be exposed through research.

4.5.1 Survey

We recruited participants through emails to two public mailing lists of ODK developers[1] and community[2]. The survey was open for a period of five weeks. We expect that the survey took around 30 minutes to complete. We asked participants to provide answers corresponding to a single deployment that they were most comfortable answering questions about, even if they had been involved in multiple projects that used ODK. We did not collect demographic information such as gender, race, or citizenship. We received 56 submissions. All participants reported that they use mobile devices for data collection and use at least one ODK tool.

Survey questions addressed a variety of topics including: type of data; domain of data; size and length of deployment; defenses considered or used; security concerns; incidents of

[1]`https://groups.google.com/forum/#!forum/opendatakit-developers`

[2]`https://groups.google.com/forum/#!forum/opendatakit`

data or device loss, theft, and/or compromise.

4.5.2 Interviews with ODK Users

To more fully understand into specific aspects of deployments, we conducted in-depth follow-up interviews with survey participants. 33 of the 56 survey respondents submitted their email address, giving us permission to follow up with them for a phone interview. We contacted all of these people via email to set up interviews. We interviewed ten of these people 33. Most of the people interviewed were in the "deployment architect" role, but some may take on more than one role. We chose to focus on these stakeholders, rather than beneficiaries, enumerators, or donors, because deployment architects typically have both a broad view of the deployment and direct involvement.

Our interviews were wide-ranging. We began with a script and had a list of questions we hoped to get answered. As the responses were free form, however, at times the order of questions changed or followed a segue specific to a previous answer. Topics covered included: the participant's role in the deployment, stakeholders or other people involved in the deployment, the purpose and organization of the deployment, what type of data is collected and whether any of it might be considered sensitive, and possible security or privacy concerns or other issues that might have arisen during the deployment. We also elicited from participants their attitudes toward the defenses described in Section 4.4. These one-hour interviews were conducted via Skype or phone and were audio recorded with participants' consent. Participants included two women and eight men. The deployments spanned several data domains (some more than one): two agricultural, five medical, seven humanitarian, and five in other domains. Nine of these deployments were in developing regions, recalling the observation in Chapter 1 that ICTD is relevant to low-resource settings beyond those in the developing world.

Interview analysis Two researchers were present for almost all interviews and took written notes, which they shared with other group members. Other researchers listened to some of these interviews, took additional notes, and looked at the notes before meeting as a group. As a group, we did an affinity diagramming exercise to come up with a list of themes. The same two researchers independently listened to the recordings and took structured notes based on the themes that were identified in the affinity diagramming exercise. These notes were then checked for agreement before reporting any results. Note that some participants' opinions may have been inconsistent within an interview—these inconsistencies were noted when observed, and interviews were revisited when researchers' notes were inconsistent to determine whether the researchers disagreed or the participant had mixed feelings about a specific topic.

4.6 Survey Findings

Before designing our in-depth interviews, we conducted an exploratory survey. Its findings helped us formulate topics and participants for further investigation in interviews.

Overview Most of the 56 respondents collected data in at least one of three data domains: medical (21), humanitarian (22), and agricultural (18). Many categorized their data in multiple ways: 25/56 collect data in more than one domain. A majority (40/56) of respondents reported using paper for data collection in the past. As we discuss in subsequent sections, the switch from paper to digital data collection may influence how people think about data security.

Sensitive data We expected to find a correlation between the domain about which data is collected and whether that data is considered sensitive by the survey respondent. Instead, we find that respondents *across* data domains consider some of the data they collect to be sensitive. Of 55 respondents who answered the question, 36 reported collecting sensitive data, spread across data domains: 16/21 in medical, 18/22 in humanitarian, and 11/18 in agricultural. This is a cogent reminder that the security of a tool should not be contingent upon its intended domain.

Security risks and incidents Our survey asked respondents about whether their deployments had encountered particular security or other incidents, such as lost or stolen data or devices. 17 respondents reported lost (14) and/or stolen (9) devices. Though a lost or stolen device could result in compromised data (since there are ways to access the data once one has access to the physical device, as discussed in Section 4.4), respondents did not necessarily equate device loss with data compromise: only 7 of the 17 respondents who reported lost/stolen devices also reported that data had been leaked, stolen, or that they didn't know. This mental model may be reasonable—for example, hardware thieves are not necessarily seeking the data on the device—or it may represent a misconception in some cases. One of the goals of our subsequent interviews was thus to learn more about these kinds of incidents, and participants' perceptions of them.

When we asked explicitly about lost or compromised data—which represents direct knowledge of a security incident—9 respondents reported that they had lost data, 2 reported a data leak, and 2 reported stolen data. These results were interesting to us because they suggest that ODK deployments *do* face at least some level of security threats in practice. We investigate these issues in more depth in our interviews.

We also find that 29/56 respondents report that devices are shared by more than one person in their deployment. This is a difference between common usage models outside of low-resource settings, where it might be expected that each user has their own device. Device sharing can pose a risk in some situations because it means that data collected by one individual could be available to other individuals or that the device's behavior (e.g., security settings) might change between users in unsuspecting ways.

4.7 Interview Findings

We now turn to our semi-structured interviews. These interviews allowed us to dive more deeply into issues raised in the survey, to provide concrete ICTD-specific perspectives to our threat model, and to gain a more complete understanding of participant attitudes toward security.

4.7.1 Concrete Security Goals

Our survey results in Section 4.6 found that many respondents collect sensitive data (including 8/10 of our interview participants), and our security analysis in Section 4.4 surfaced several interpretations of what “sensitive” might mean. We use our interviews to understand more concretely what our participants mean by “sensitive” as well as their broader security goals. We find that data availability and data integrity are explicit, prominent goals for ODK deployment architects, whereas confidentiality is not; however, when prompted to threat model, ODK deployment architects do identify reasons for which confidentiality can be important in ICTD contexts.

Data loss (availability) For several participants (6/10), a main concern related to the potential derailment of their deployment was data loss. To ODK deployment architects, data loss refers to collected data becoming permanently *unavailable* (and not to the computer security concept of data loss prevention, or the exfiltration of confidential data). P1, when asked about the greatest threat that could derail a deployment, responded “far and away it’s data loss. It’s just losing the data somehow”, going on to say “I don’t worry about somebody getting a copy of the data, I just worry about getting the original data from the remote enumerator into a centralized database.”

In one case the possibility of data loss outweighed the perceived value of security features like encryption, which would support higher *confidentiality* of data:

> P2: [Encryption] did cause us some problems and that’s why we didn’t continue it .... You try to submit some data, some of them get lost along the way somehow.

Erroneous data (integrity) Another significant security goal that deployment architects identified was the protection against the entry of falsified data by enumerators. 6/10 participants indicated that they know they have received false data in the past. This goal is a specific example of a data *integrity* goal, against a specific class of adversaries. From the interviews, we learned that enumerators might fabricate data for several reasons, such as avoiding travel to interview locations or shortening interviews by answering “no” to questions that lead to long subsections of a form. P5: “Enumerator fatigue is just as real as [beneficiary] fatigue.”

Exploited data (confidentiality) Compared to data loss and erroneous data, participants seemed significantly less concerned about data *confidentiality*. When asked to threat model possible ways in which unauthorized parties might access data and what their goals might be, participants identified a number of *hypothetical* consequences of data breaches:

- Loss of job, e.g., for beneficiaries
- Loss of life or threats of physical harm
- Theft or looting, e.g., knowing a place is vulnerable
- Embarrassment in front of donors

Several of these consequences fit under the broader umbrella of avoiding harm to beneficiaries—a risk that ODK development architects seemed particularly concerned about, when asked to ponder the impact of data disclosures.

Although the impact of a data breach can be greater than the impact of lost or erroneous data, the overall risk of such a breach might be less (since, presently, adversaries do not seem to be intentionally trying to breach the confidentiality of the data collected in these deployments). Deployment architects we interviewed seemed to have informally come to this conclusion. Consequently, participants did not consider the potential of data breaches to warrant additional protection beyond their current practices. Moreover, we asked all participants if there was any data that they were interested in but did not collect for computer security reasons; the answer was universally no. Despite the low risk, the damaging effect of data breaches is also not purely hypothetical; P3 reported that a beneficiary lost a job when similar data collected in a previous (non-ODK) deployment was reported to their supervisor.

4.7.2 How Threats Could Manifest

Having now mapped from the abstract security goals in Section 4.4 to concrete instantiations, we turn to studying concrete threats against ODK deployments.

Hardware theft (or loss) Theft of devices was one of the threats that participants thought was the most likely to happen in their deployment. This threat is not hypothetical. Recalling Section 4.6, 17/56 survey respondents reported lost or stolen devices. In our interviews, three reported having devices stolen, and one participant seemed surprised that no devices had yet been stolen in their deployment.

However, echoing our survey findings, data *confidentiality* was not thought of as the primary concern if a device was stolen or lost. Instead, security of physical devices was approached as a separate issue from data security by most participants. For example, one participant indicated:

> P2: We're hoping it's just about the hardware, that's fine. But I don't think it could be an issue about the data inside the tablet. . . . It's kind of fine, like "take it, reset it, don't look at the data, and enjoy the tablet."

Despite the known potential for device loss or theft, not all participants considered device loss or theft a serious risk; some thought that the devices were expendable.

Enumerators as adversaries Enumerators are recruited using a variety of strategies, ranging from organization employees to anonymous volunteers with no organizational affiliation. Whereas most enumerators likely have no ill intent, participants identified a number of ways that an enumerator could become an adversary:

- Selling data being or coerced to leak data
- Fabricating data to avoid some part of their job (e.g., travel to a possibly dangerous location, fill out a long boring questionnaire, ask uncomfortable questions)
- Make honest mistakes with data entry

- Accidentally download malware or use excessive amounts of data for non-work activities
- Not caring for the hardware properly/sufficiently

All of the people we talked to have concerns about the veracity of the data being collected by these enumerators, which relates to the data integrity goal. Six were aware of enumerators entering fake data in the past. In at least one case, an enumerator was fired upon clear evidence that he knowingly and intentionally submitted fake data.

Participants noted both technical and non-technical methods that an enumerator could use to compromise the privacy of beneficiaries. There was skepticism that enumerators have the skill to mount even modestly sophisticated technical attacks. However, P3 noted the feasibility of non-technical attacks. The easiest way to gain access to the data might simply be to talk to the enumerator that conducted the interview: "it would be much easier to bribe or go and see an enumerator and offer him a beer." A key lesson is threats may exist regardless of the technical defenses.

Weak vs. powerful adversaries Most of the attacks we have discussed so far would be carried out by adversaries with limited technical skill and resources. Although these weak adversaries can pose threats, a powerful and motivated adversary can introduce a wider range of threats and potential attacks. For example, if an attacker does not know how to copy data off of a device, the device may be much safer than paper, which can be read by anyone. For a stronger adversary with technical skill, however, digital data could be duplicated without leaving evidence of a data breach.

The most explicit example of a powerful potential adversary came when one respondent indicated that their data, when viewed in aggregate, could reflect a group perceived as dangerous in a negative light, stating:

> P2: We record violence that those children might have went through walking on the street. And actually it turned out that the highest perpetrator is, well I cannot mention the name now, but ... it's a very dangerous group. We still collect it on the tablet but we don't give out this information to anyone because

> it will put everyone at risk, whether us or those children, everyone in the program will get in trouble if we give out who is the highest percentage of the perpetrator ... We only give it to the UN agencies if they request it, and then we keep it in conversation, we don't actually email even this information, because if it gets intercepted, again, everyone would be in trouble. So it's not stopping us from collecting the data, it's just sharing the data becomes trickier.

In general, even if they had given consideration to more powerful adversaries, they did not seem to have ideas of how those adversaries might compromise their system.

4.7.3 Mitigations

We now explore the steps participants take or might take to mitigate computer security risks.

Defending against enumerators faking data While most enumerators are likely trustworthy, Section 4.7.2 identified untrustworthy enumerators as a real threat. 6/10 respondents indicated that they include explicit checks into forms or look for data abnormalities to detect possible fabricated data. These checks include: (1) GPS readings to ensure the enumerator was in an appropriate location when completing the form (2/10), (2) timestamps to measure survey completion time (2/10), and (3) requiring enumerators to take photographs of relevant locations (2/10). In some cases enumerators are informed these checks are monitoring their actions, while in other cases they are not.

However, others noted that these checks do not come without additional costs. In some deployments, there might be a desire not to collect GPS coordinates due to sensitivity or technical constraints (e.g., the enumerators are surveying a threatened population or because it takes too long to register a GPS signal). A broader lesson, well-known in the computer security literature, is that computer security defenses can come with a cost, and that it can be challenging to balance between the benefits of the defense and the additional costs; in this case, for example, collecting GPS coordinates for all deployments could result in violating the privacy of enumerators (since their precise locations would be known) or harming beneficiaries (if the locations of the beneficiaries are sensitive, and the data were ever to be exposed).

Defending against non-prescribed device use The security of devices can be compromised by malware or non-prescribed uses that could impact the deployment, e.g., by transmitting data to an external source or consuming a data quota. Organizations were aware of this threat and took several precautions to limit certain aspects of device use. 4/10 respondents reported that they employed defenses at the level of the mobile device software, such as an app that limits the functionality supported by a device. For example, the deployment devices might only allow the ODK app and a GPS app to be opened without an administrator password.

However, some participants provided reasons why they would not completely lock down devices. One participant made exceptions based on the context of individual enumerators' duties. Those going into a conflict region, for instance, were given devices that were not as locked down, including access to phone and email apps, to keep them useful if they were to be in danger and require a phone:

> P3: I couldn't see myself limiting them from the benefits they could get from the tablet in case they were in this kind of [dangerous] situation. Meaning having access to phone, having access to their emails.

The existence of these exceptions speaks to the tradeoff mentioned earlier: incorporating computer security defenses can have negative consequences, and the benefits of security defenses may not always outweigh those consequences.

Protecting devices Since the risk of losing or breaking a device is real, and since the devices are physically under the control of enumerators, some participants employed mechanisms to help enumerators physically protect their devices. A strategy expressed by at least three participants was to confer liability of the tablet (e.g., cost) to the enumerators, believing that this would cause enumerators to taking greater care with the tablets. Some gave the participants protective cases along with devices, used plastic containers to transport devices, or locked devices in cabinets.

Participants also described what they do after a device has gone missing. One survey respondent mentioned tracking down the device based on GPS location. Others used the phone's remote wipe capability, if the device was able to connect to the Internet. The use of remote wipe suggests some concern over protecting the confidentiality of collected data though, as noted in Section 4.7.1, confidentiality did not emerge as a goal on the forefront of participants' minds. Indeed, many of other precautions employed by participants to protect devices (e.g., making enumerators financially responsible or locking devices in cabinets) speak more to protecting the devices themselves, not protecting their data.

The need to protect devices was not, however, universally recognized, with multiple participants downplaying the importance of protecting devices. One such participant reported zero devices stolen and only a single device damaged. P1 reported never having a device stolen, and reported believing that the "the job itself is more valuable" to enumerators than tablets, going on to say of device theft: "I think that it's a baseless worry".

Backend choices Although most of the threats discussed so far pertain to data while on a mobile device, the security of data is a consideration when aggregated as well. No participants indicated that they are concerned about accidental or malicious modification of data once it is stored centrally. Participants expressed varying degrees of understanding about the security guarantees of their data backends. P6 was aware of avenues for data accessed once the data was on a server that could have implications for confidentiality. P3 reported that they would have liked to determine how frequently their partners were viewing the data, as well as to know if the data had been downloaded.

Data is moved to at least one external hosting site in 8/10 deployments. One participant indicated familiarity with the security guarantees of their hosting company:

P2: They say that everything is secure and that their servers... [are] underground

> [providing] maximum security, no one can infiltrate their data.

Moreover, in this case the participant seemed to delegate security decisions to the hosting company, trusting the hosting company to be secure rather than encrypting their data. Delegation of security responsibilities to other entities may, however, not always be warranted.

Security through obscurity Some participants felt that digital data collection provides some measure of security through obfuscation. To access data on the device requires technical skills and/or knowledge of where and how data is internally represented. An adversary with these skills might be able to duplicate or modify collected data without leaving any trace of their activities. Generally, however, participants were not concerned about this type of attack. For example:

> P5: If someone's really interested in the content, it's easier to steal a stack of papers than it is to steal . . . the technical maze you need to actually make sense of the data. So relative to someone on the ground being able to steal the data content, tablets are much more secure than paper.

This might have been true for the type of adversaries they were most concerned about, but we point out that these technical skills (as evidenced in an earlier section) can be common depending on the context and the adversary. State actors would not balk at the technical capacity required to compromise a mobile device.

Attitudes toward proposed defenses In Section 4.4.2 we discussed several possible defensive directions, including defenses already available within ODK, additional best practices, and new directions. Although no one used encryption, the potential value of encryption was widely recognized. Participants did identify challenges to using encryption. For example, P2 indicated debugging data loss was more difficult when encrypted: "at what point something went wrong—I was not able to figure that out".

As a negative result, and contrary to our expectations, no other defense was seen as potentially useful by a majority of participants. Concerns with these defenses were based on (1) (perceived) difficulty of setup, (2) difficulty of use (for enumerators), (3) sufficiency for solving the problem, and (4) monetary cost. As a concrete example, P6 believed that password-based protection would not be effective due to the practices of enumerators. "[Passwords] might not work with people because. . . I will give it to you because I know you." Geo-based security would not work for P3 because there is no GPS data available for the sites they are surveying, making GPS impossible to configure. P5 reported that "sometimes the GPS just doesn't work on these devices."

4.7.4 Broader Context

Finally, stepping back, we discuss several factors participants surfaced that influence their knowledge, mental models, and actions surrounding computer security.

Roles and responsibilities An important factor in what technical defenses are considered or implemented is how much responsibility the deployment architect takes for determining the appropriate security mechanisms.

> P7: You know the company in their mind, they would keep saying, "Well, after the pilot is done, you're going to bring the server back to our data center, right?" And I kept reminding them that, look, "Are you guys experts in security? What if someone really got interested in the entire data set? What if they hacked in? Are your IT guys savvy enough?" . . . It was sort of a process, we were trying to encourage them that the cloud is a [more] secure environment for them.

Another participant cited a lack of technical expertise as an explanation for their decisions to make weaker security-related decisions:

> P8: And the local server also has several passwords, and the password that I'm using for the local servers is probably not as secure as it should be. I'm not

> really an IT guy, and as you may know setting up a local server is complex and a bit finicky. It's very easy that it doesn't work, and it's hard in my experience troubleshooting getting it working right, so I've tried to keep things as simple as possible, and it works but it could be more secure.

As we discuss further in Section 5.4, the dispersal of responsibility for computer security decisions among different stakeholders in a deployment, and among people with different degrees of technical expertise, affects the security-related decisions and tradeoffs that are made.

Ethics board considerations Another group that may (or may not) be responsible for enforcing secure practices is the Institutional Review Board (IRB). Most participants considered ethical issues related to their deployments, and three reported experience with an IRB. These three participants had experiences with IRBs in more than a single country and stated that the IRB process and requirements varied widely between countries:

> P1: There are not universal requirements for the IRB . . . It's kind of wild west stuff actually. There's no universal requirements . . . They will put requirements that are specific to that piece of research so it depends on very much on the area.

This variation may be desirable: P3 mentioned that an in-country IRB supported work that was relevant and acceptable in the local context but that may have seemed problematic to foreign IRBs. However, weak IRB requirements can also have negative security consequences for deployments. Since the IRB is in a role of requiring compliance with its policies, deployment architects may defer to those requirements. IRBs, however, may lack the technical expertise (or otherwise fail to) require specific security practices:

> P1: [Encryption is] something [the IRB] should be requiring and . . . just lack the technical sophistication to ask for it.

As the technical contractor hired for the deployment, this participant did not consider themselves in a position to impose technical requirements on the study, deferring instead to the IRB's (weaker) requirements.

Community privacy and security norms Although our goal was not to understand the subtleties of the communities in which ODK is deployed, local norms may play an important role in threat modeling. For example, although knowledge of what method of birth control a woman uses may not be particularly sensitive data in some places, such data could be considered significantly more sensitive in places with more conservative values. Similarly, some types of data may be considered less sensitive in local cultures than in the (different) culture of a deployment architect. For example, one participant was surprised by the amount of information shared by a collaborating organization:

> P4: The privacy concerns of the schools from my experience are not particularly strong. For example, something that I would consider to be sensitive is they have ... information about either special needs or poor households ... When we've requested summary statistics from them, we've often received a lot more details than I would expect them to be comfortable sharing. ... In general I think, in the rural areas where we mostly work, a lot of these things are kind of treated as common knowledge. Within the village everybody knows who's poor, everybody knows if you have some sort of special needs, so I don't think it's really on the forefront of their minds.

Sensitive groups Five participants considered data they collected about vulnerable groups—such as children, women, refugees, and victims of violence—especially sensitive. One participant voiced this concern emphatically, suggesting that the vulnerability of the population was a larger consideration than the particular type of information collected:

> P5: The most sensitive part about it I think that would jump out to an external eye is just the fact that you're working with a vulnerable population, and that

> population being children. Kind of full stop there. … Yeah, you're collecting data about their weight and their age and stuff, which ultimately isn't sensitive, but you're collecting data about children.

Historical context: paper forms When the technology used in a deployment changes, mental models surrounding that new technology are influenced by mental models about the old technology. In this case, many (9/10) of the participants had previously employed paper forms for data collection, before switching to ODK for digital data collection. In some cases participants still use paper forms for some parts of their data collection. Though there are many axes along which to compare these two technologies, we focus specifically on security issues here. Participants had mixed views about the security of digital data collection as it compares to the security of paper data collection and gave examples both of how they believe digital may be more secure than paper and of how paper may be more secure than digital.

Several key security *advantages* that participants mentioned of digital over paper are based in the fact that data is uploaded to a server (when a network connection is available). This syncing improves data availability—reducing the amount of data lost when a device is lost—and data integrity—allowing rapid feedback to enumerators to improve the quality of their data. There was also a perception that digital data collection improved the confidentiality of data, because it increases the technical barrier to reading data from a device than from a paper form:

> P1: Those surveys that were done on paper were openly readable to anybody who had access to the paper. And their privacy is better served by putting the data into a format that is inaccessible. Our enumerators collect all this data, but as soon as they collect it, it's locked away from them. So they can't share it with each other in any way. And they can't review it. They can't do anything. They can record it, but then all they have is their memory of it.

This perception reflects the previously discussed "security through obscurity" mental model exhibited by several participants—that the barrier of technical knowledge is sufficient to protect the data. In the face of a moderately technical adversary, however, the threat to confidentiality may be greater, since that data can be easily queried (unlike searching through many paper forms). Participants generally did not consider such adversaries, however.

Increased confidentiality can also come from the ability to employ technical means to protect digital data:

> P1: The digital database provides a kind of guarantee of confidentiality if only because it's controllable and inaccessible and transportable in a way that is controllable. You can password protect every part of it.

On the other hand, a key security *disadvantage* of digital data collection that participants mentioned was that it affords the collection of more, and more sensitive, information:

> P1: The only thing that is specifically unique to doing digital data collection is that you have more identifying information, like you have the GPS coordinates of people's homes and you have the times and dates when they were there. ... And you could have photographs and audio recordings and all kinds of things ... those things are unique to digital data collection.

Finally, there are differences in security perceptions on the part of the people from whom data is collected. One participant mentioned that beneficiaries are (perhaps incorrectly) more trusting of tablet- than paper-based data collection:

> P2: [When] you use the tablets... they feel a bit more safe than when you're using the paper forms. ... They ... trust you more when you're using those tablets because they also assume that it's going to be more secure than carrying around a bunch of paper forms ... But actually if they know how the tablets also work... technically it's still as insecure as paper methods.

4.8 Discussion

Having presented our threat model, survey results, and interview results, we step back and reflect upon their broader implications. These implications are in addition to more specific lessons and recommendations mentioned previously.

4.8.1 Broader Considerations

Diversity of stakeholders and views on security As in other ICTD contexts, our results surface the importance of considering the full spectrum of stakeholders, who may each have different perspectives on computer security. (Indeed, as we consider in our threat modeling process, some stakeholders may also become adversaries.) We summarize several previous examples to underscore the importance of considering this diversity. For example, ethics boards have a sense of breadth that comes from their exposure to many different projects, but are not necessarily well-versed in specific technical best practices. Meanwhile, the data being collected ultimately comes from beneficiaries, who may have different perceptions about the sensitivity of their own data. External parties do not necessarily understand local context, while locals may not realize how information could be misused outside of that context. These differing perspectives must be carefully considered for each deployment. The diversity of stakeholders and stakeholder views on computer security means that there may not be a "one size fits all" solution for computer security for ICTD systems.

Challenges with diffused responsibility A consequence of the diversity of roles and responsibilities within a deployment, also surfaced in Section 4.7, is a perceived (or actual) dispersal of responsibility for security. This can lead to an environment where no one feels they are able to intervene with what they consider best practices. This surfaced in multiple ways. For example, one participant was thinking about security but felt unable to act because their development contract did not ask for security defenses. As another example, in their description of how their system uses a cloud hosting company, a participant delegated security responsibilities to the hosting company rather than consider the system holistically. Even if security decisions are consolidated to a designated person within an organization, an additional challenge is that many actors may still have a role in implementing the chosen security defenses, ranging from enumerators to development architects.

Considerations for threat modeling Deployments are more likely to be secure if, before data collection begins, an organization considers how data might be used and who might want it. As part of our interviews, we invited participants to create threat models for their deployments. Some had already begun this process, but only one had done it formally. We believe that this process is valuable, because even if no new or realistic threats are uncovered, it is important to make security choices grounded in a thorough understanding of the tradeoffs rather than in an ad hoc manner.

A common theme in our interviews was the perception that the relative technical sophistication required to access digitally collected data made it more secure. Though many of the realistic threats and adversaries considered by our participants may indeed be thwarted by the need for technical expertise, we caution deployment architects to consider more sophisticated adversaries as well.

In particular, technologies, threats, and adversaries may change over time, so threat models must be periodically reevaluated. Collected data may be retained for a long time, and in that time it is possible that new attacks will emerge that make it easier to access that data and/or that the data may be used or combined in unanticipated ways. Even if an attacker's capability does not evolve, the value of a beneficiary's data may change over time (e.g., as a child grows up and enters politics), increasing the willingness of an attacker to put forth technical effort to carry out an attack. Section 4.7 discussed how a transition from paper to digital data collection has already affected threat models, emphasizing that threat modeling must be an ongoing process.

ICTD security can leverage traditional security In studying computer security for data collection systems in ICTD, we find that, broadly speaking, the challenges to implementing computer security in an ICTD context echo challenges that are well known in non-ICTD contexts. For example, both contexts face tradeoffs when attempting to integrate security with other (usability or functionality) goals, and both contexts can benefit from employing computer security best practices. One difference, however, is that there have been few high profile attacks on ICTD data, and hence ICTD deployments have not felt the same adversarial pressure as other technology domains. We posit that if high profile attacks do emerge, they will transform organizational attitudes toward the likelihood of future external threats.

4.8.2 Recommendations for System Designers

Finally, we step back and make recommendations for the designers of systems like ODK.

Implement defenses to fit current workflows Since users of a system must make practical tradeoffs, it is important to design defenses and other security features so that they fit into existing workflows. Successfully integrating into workflows is non-trivial. It is a central issue in Chapter 5. Designing systems that are both secure and usable is a central challenge in security research. Security and ICTD is not free from these challenges. For example, in Section 4.7 we found that no participants enable encryption, despite abstractly finding it valuable. Often this decision was made consciously, not accidentally—those who experimented with encryption chose not to use it when they lost data or because they found it made debugging more difficult. In other words, if ODK simply made encryption the default, that would not necessarily increase its use and may harm other deployment goals (e.g., data availability). Instead, features like encryption must be designed in a way that also supports other deployment goals (echoing existing lessons learned in the computer security community, e.g., [87]).

Support auditing of device use and data Increasing the extent to which systems can be audited would address concerns about data integrity (Sections 4.7.1 and 4.7.2). Logging mechanisms could detect non-prescribed use (Section 4.7.3) but allow non-prescribed actions, like phone calls, in emergencies, compared to phone locking applications which cannot make exceptions without a password. Records of when and how often data was viewed, both on the device and in the cloud, can reveal access patterns that might indicate inappropriate curiosity or malicious intent. Similarly, one participant suggested that recording when a screenshot was taken would be useful to indicate that data may have been inappropriately captured for distribution.

Consider the broader technical ecosystem In addition to considering human factors of a threat model (e.g., different perspectives on security), it is important to consider the broader technical ecosystem in which an application may be used. For example, malicious applications may be installed on the same device as a digital data collection application like ODK, suggesting that designers should be cautious about data they write to world-readable locations on the device. Additionally, system designers may rely on external components for certain functionality: for example, QR codes—as used by one of our participants—may be read by an external application; indeed, ODK developers recommend using a third-party application to scan QR codes. System designers must include these external components as part of their threat models. For example, some QR code applications may transmit QR codes or GPS coordinates to their backend systems, which may violate the data flow and data privacy expectations of an ODK deployment architect. Consequently, it may be preferable to implement certain functionality directly into a system rather than relying on (possibly untrusted) external components.

4.9 Conclusion

Digital data collection is an important activity for many organizations in the developing world. We focused on ODK as a widespread digital data collection platform and conduct a computer security threat modeling exercise to evaluate attacks that could target ODK deployments. We conducted a survey and interviews with organizations using ODK to understand what threat models are considered in the field. Leveraging our threat model, survey, and interview results, we explore the challenges of computer security in digital data collection in an ICTD context and make recommendations to organizations seeking to keep their data secure.

Chapter 5

DUCES: A FRAMEWORK FOR CHARACTERIZING AND SIMPLIFYING MOBILE DEPLOYMENTS IN LOW-RESOURCE SETTINGS

Even in the most capable hands, technology is never a complete solution. Understanding how to effectively use tools like chose described in Chapters 2 and 3 is as much of a challenge as creating the tools themselves. Devising effective workflows and deciding how to incorporate technology is a significant challenge. In this chapter I describe DUCES, a conceptual framework for characterizing mobile deployments along five axes of design. DUCES allows organizations to better understand deployment requirements and simplify decisions regarding workflows. DUCES focuses on the workflow's **D**ata flow, **U**ser interface, **C**onnectivity model, **E**dit mode, and **S**erver requirements.

This chapter originally appeared at the 2015 ACM Symposium for Computing on Development (DEV '15) [76].

5.1 Introduction

Mobile devices are deployed in many low-resource settings for data-focused applications. Creating these applications is nontrivial, consuming considerable time and resources [42]. Successful custom-built deployments can require years of iterative refinements to work out stable technical architectures [61]. Smaller scale deployments often do not involve developers, making reasoning about technology difficult. We present the DUCES framework, which can be used to deepen understanding of a deployment and its requirements as well as to highlight which requirements are most challenging technically. If these can be altered in a way that simplifies the architecture, the deployment will become more sustainable without external expertise.

We have observed that a number of common paradigms exist in data-focused mobile deployments conducted by groups in low-resource settings. Based on our experience, we characterize these deployments along five axes of design: whether the **D**ata flow is unidirectional or bidirectional; whether the **U**ser interface (UI) is form-based or non-form-based; if **C**onnectivity is required to function; if **E**dits are non-transactional or transactional; and if the supporting **S**erver is merely a data repository or if it encapsulates logic. Using technology always presents challenges. It is easy to argue for simplification, but developing intuitions around how to do so can take years of experience and can be specific to a single technology. The DUCES framework provides a way to approach simplification that is generalizable to a wide range of scenarios and tools.

DUCES is aimed at small and medium-sized organizations seeking to leverage mobile technology in low-resource settings. These organizations generally do not have the resources to devise a custom technical solution. They are not creating new technical frameworks and they do not have a developer on staff. They are seeking to build on top of existing solutions to leverage mobile technology. Such organizations frequently face difficulties when trying to reason about diverse requirements and their implications [17]. In these organizations, deployment architects are generally not developers themselves. This frequently makes the implications of requirements opaque. Many deployment architects lack even basic intuitions about what is easy and what is hard. For example, supporting two languages is a fundamentally different problem than working in the absence of an internet connection. Sending SMS reminders automatically based on HIV status is more challenging than capturing GPS data. Organizations frequently treat all requirements as equal, even though disconnected operation or server-side automation might complicate the deployment by orders of magnitude.

In this chapter we explore five case studies of mobile deployments that leverage technology in different ways. We analyze these deployments to gain a comprehensive understanding of the various technical requirements that exist in mobile workflows in low-resource settings.

The contributions of this chapter are to formalize a framework for understanding and simplifying these mobile-based workflows. This framework, which we refer to as DUCES, elucidates characteristics and intuitions that are latent in many data-focused mobile apps, including those in high-resource environments, but that take on increased importance in low-resource settings. Using DUCES, organizations can identify early in their process what components are likely to require outside technical support and what is able to be accomplished in-house.

The paper is structured as follows. In Section 5.2 we outline five case studies of mobile deployments in low-resource settings. We summarize the requirements and goals of each case study. In Section 5.3 we describe the five axes of design that define the DUCES framework. We revisit each case study, exploring how the requirements of the deployment impacted the deployment architecture. In Section 5.4 we discuss how the traits of our framework highlight fundamental challenges that exist in mobile deployment architectures, how certain features are in tension with one another, and the ramifications of these considerations on mobile deployments. We revisit each case study to describe how DUCES was used to simplify or could potentially simplify each architecture. We close by discussing the ramifications this work has on organizations deploying mobile apps in low-resource settings.

5.2 *Case Studies*

The goals of a mobile deployment define the technical requirements. This is not always a straight forward process. In this section we present five data-focused mobile deployments to serve as case studies. They were chosen to provide a broad sample of requirements. Characterizing them through the lens of the DUCES framework lends insight into what sorts of technical solutions are appropriate given the constraints of the deployment. Three of these case studies have been conducted by the authors, allowing insight into how DUCES was used during development of the deployment.[1] All have been deployed and used in the field. Two case studies are based on published literature.

[1]Two of these three deployments are presented here for the first time.

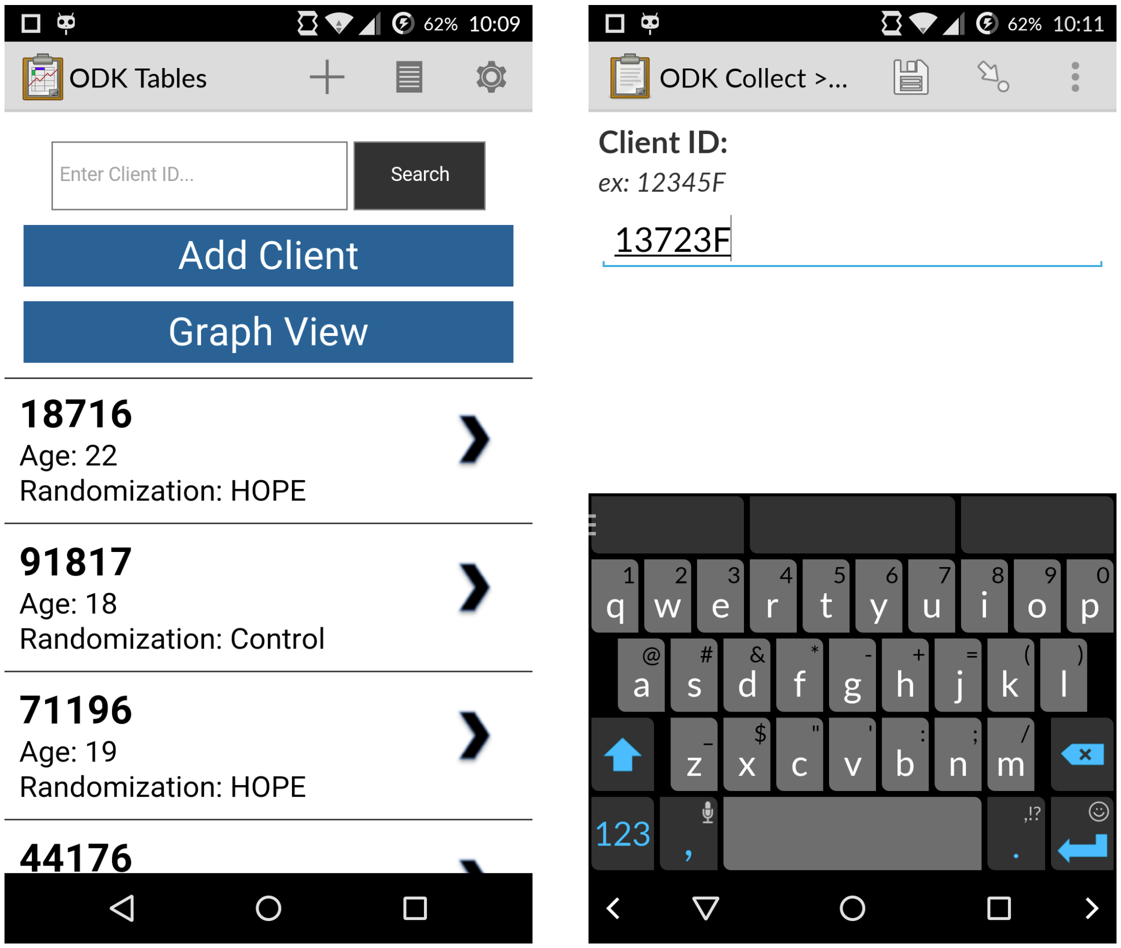
5.2.1 Longitudinal HIV Study

The first deployment is support for a study of HIV discordant couples in Kenya [79], as previously discussed in Chapter 2. The study itself was designed by global health researchers in order to longitudinally monitor couples where one partner is HIV-positive and the other is not. Participants are screened, at which point data is collected, and at time points in the future additional sets of data are collected. For both screening and follow ups, participants are administered a survey on a mobile device. Different data is collected at different time points about male and female participants. A particular form is administered to a participant based on the time they have been enrolled in the study and their gender. This model of an entry form with follow up forms is common in research studies [36].

Each day study coordinators perform basic analysis and create a list of participants that require follow up. This data includes the subject's unique identifier and the form that is required to be completed. Enumerators are hired to take this information into the field, locate the subject, and complete the specified form. They are equipped with mobile devices that render the forms and provide a simple app-like user interface. Upon opening the app, enumerators arrive at a home screen and are trained that they can screen a new participant, perform a follow up interview, or submit collected data to the server. Sample screens are shown in Figure 5.1.

5.2.2 Tuberculosis Test Results

In 2009 a digital form-based workflow was deployed in Lima, Peru to digitize tuberculosis (TB) test results [5]. Sputum smears are collected at local health centers and the test results are written in a ledger. Enumerators visit these health centers with personal digital assistants (PDAs) that are equipped with digital forms. The forms have been designed to collect the information that has been written in the ledger, and enumerators transcribe the contents of the ledgers to the digital forms on the PDA. Upon returning to the central office, the data is uploaded from the PDAs to an Oracle databased managed by Partners in Health,



18716
Age: 22
Randomization: HOPE

91817
Age: 18
Randomization: Control

71196
Age: 19
Randomization: HOPE

44176

Client ID:
ex: 12345F
13723F

Figure 5.1: Examples of the mobile app for the HIV study. The participant list supports prepopulation of identifiers (left). If not available, the identifier can be entered manually (right).

a non-governmental organization (NGO) operating in Lima. Data upload takes place over the internet using the open database connectivity (ODBC) standard. The study designers also extended the database to include automated processing of the data as well as web pages that allow for summaries of the data and provide data quality checks. With this workflow, processing times for samples were greatly improved.

5.2.3 Supply Chains Using Mobile Phones

Mobile devices have been deployed to improve the performance of rural supply chains in resource-constrained environments [70]. For this deployment an organization (Logistimo) noticed that stock outs were occurring at health centers in large part due to poor information management and communication. They created a Java application for feature phones that allows pharmacists to enter stock-related data, including the sale of items and stock counts. This data is transmitted to a server via a cellular data connection or SMS. The server component processes the data and removes duplicates that have arisen due to network errors. The server is also responsible for sending alerts to supervisors via SMS or voice calls. In addition, the server provides a "bulletin board" web application that shows streaming information about the state of the supply chain. Synthesizing data in this way created actionable items that resulted in a drastic increase of availability of vaccines at local health centers.

5.2.4 Chimpanzee Monitoring

As mentioned in Chapter 2, the Jane Goodall Institute (JGI) has employed a complex chimpanzee monitoring system for a number of years. Under the system, a ranger follows a group of chimpanzees through the forest over the course of a day with a complex paper worksheet. This activity is referred to as a "follow", and is broken into 15 minute time intervals. Data is collected about each interval. Various data is recorded, including when chimps arrive and depart, the estrus state of the female chimps, the foods consumed by the chimps, and the presence of other species. All this information is captured on a dense paper worksheet consisting of a matrix with 15 minute intervals on the y-axis and chimp identifiers on the x-axis. Arrivals and departures indicated by drawing a line in the corresponding 15 minute interval. A copy of the paper form is shown in Figure 5.2 in order to convey the density of information on the worksheet. These sheets are periodically sent to researchers that transcribe the data into a database.

One of the strengths of this model is that rangers are able to see a summary of the day's data at a glance, making it easy to visually audit data and revisit time points as the day progresses. It also facilitates non-standard data entry. For example, times are not recorded by writing an hour and a minute. Instead the ranger draws a line in the first third of the box representing 9:00 to 9:15 to show that the chimp arrived or departed between 9:00 and 9:05. We designed an application to run on a 10 inch tablet that mimics this workflow. The tablet was smaller than the normal paper form employed by the rangers and displayed data of a single 15 minute interval at a time instead of the whole day's data. Crucially, however, data is displayed as it can be edited, affording rangers similar auditing power to the original paper form. The application also uses icons to achieve the same pictographic data entry to represent time that is provided by paper. The familiar tabular structure is preserved. A comparison of the paper form and the tablet application is shown in Figure 5.2.

5.2.5 Aid Distribution

The International Federation of Red Cross and Red Crescent Societies in the Americas (IFRC) often handles aid distribution after natural disasters. They recently piloted a program where debit cards were distributed instead of physical goods. We devised a mobile system to support this workflow. The pilot took place over two days in Kingston, Jamaica, and involved 93 participants at two locations.

Registration and distribution are separated into two distinct phases. In the registration phase, beneficiaries are entered into the system using digital forms on four mobile phones. Basic data like name and address are collected. Each beneficiary is also assigned a paper card with a bar code that will later entitle them to receive their debit card. After screening

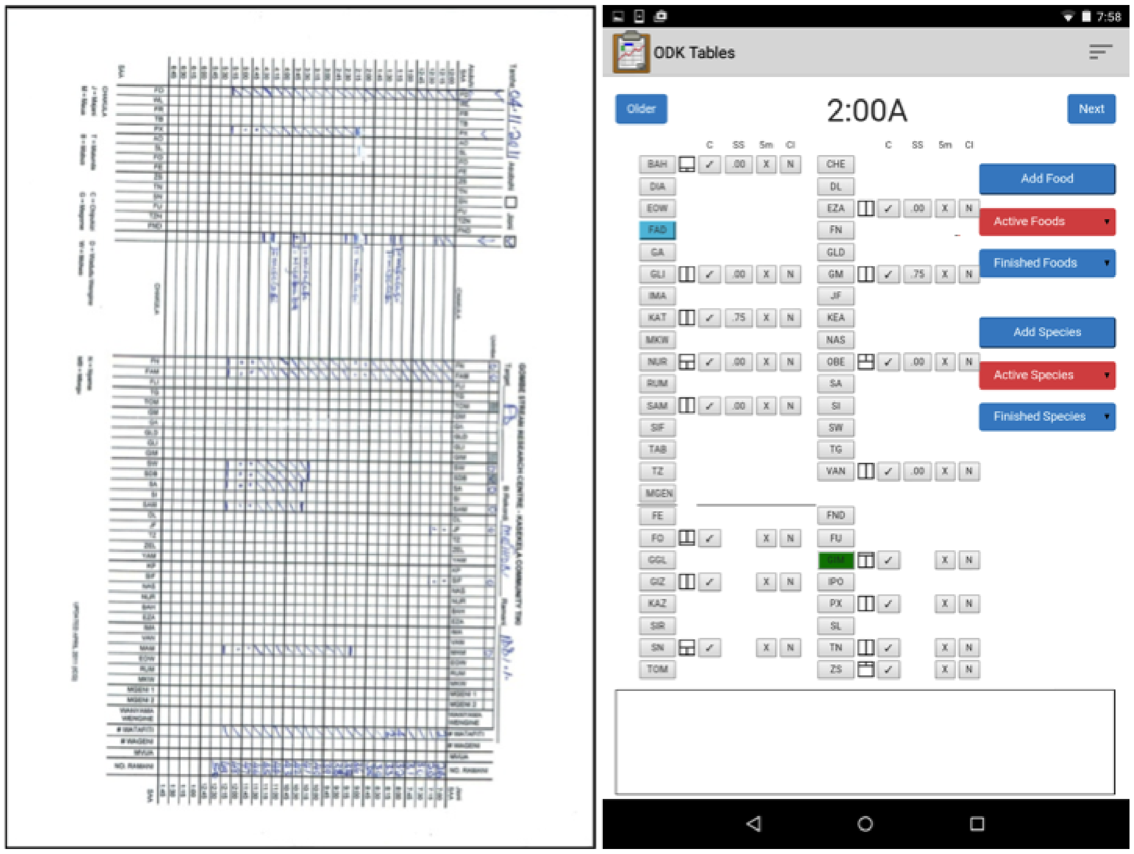

7:58
ODK Tables
Older
2:00A
Next
C SS 5m CI
Add Food
Active Foods
Finished Foods
Add Species
Active Species
Finished Species
BAH DIA EOW FAD GA GLI IMA KAT MKW NUR RUM SAM SIF TAB TZ MGEN
CHE DL EZA FN GLD GM JF KEA NAS OBE SA SI SW TG VAN
FE FO GGL GIZ KAZ SIR SN TOM
FND FU GIM IPO PX SL TN ZS

Figure 5.2: The Chimpanzee Monitoring Worksheet (left) and Application (right). The left image shows the paper worksheet employed by JGI rangers. It permits continuous review of data as it is updated. The right image shows this technique replicated in our application. This permits stylized reviews mimicking the paper-based workflow.

and before distribution, each screened patient is assigned a debit card number. Distribution occurs an hour later. During the distribution phase, beneficiaries present their beneficiary card and bar code, which is used by the mobile app to retrieve their information. After confirming that it is correct, they are presented with the cash card and marked in the database as having received the card.

5.3 The DUCES Framework

The DUCES framework provides a way to characterize mobile deployments. The framework can be applied to mobile-based workflow, including applications produced by organizations with significant resources. However, it is most useful in low-resource contexts where many simplifying assumptions appropriate in high-resource settings are inadequate. DUCES consists of five axes of design:

1. Data Flow (Unidirectional vs Bidirectional)
2. User Interface (Form-Based vs Non-Form-Based)
3. Connectivity Model (Connected vs Disconnected)
4. Edit Mode (Non-Transactional vs Transactional)
5. Server Model (Bucket-Based vs Processed)

Understanding where a deployment falls along these axes provides meaningful insight into the technical requirements of a deployment. Deployment architects without a strong technical background often lack sound intuitions surrounding mobile deployments. For example, we have seen non-technical collaborators assume that altering the text accompanying a question in a form will be as difficult as adding bidirectional data flow to an existing data entry tool. DUCES is intended to better scaffold reasoning about deployment requirements in order to prevent this sort of misunderstanding. If a deployment requires a bidirectional data flow, a data entry tool that does not support the bidirectional movement of data can be discounted out of hand. Better still, the bidirectional data flow requirement might be obviated by a slight change in deployment protocol. DUCES provides a set of primitives that can guide an understanding of technical requirements.

In the following sections, each axis of design is described in detail. The case studies from Section 5.2 are used to illustrate how DUCES can be used to describe a variety of mobile deployments.

5.3.1 Data Flow

Mobile devices are frequently used to collect and manage data, aggregating it on a server. DUCES asks if this flow of data is unidirectional or bidirectional. In other words, does collected data move only from the mobile device to the server, or does it also move from the server to the mobile device? A unidirectional data flow is simpler to implement than a bidirectional data flow, but it is also less versatile. Deployments can use each design to great effect.

Longitudinal HIV Study The HIV study is a **unidirectional** data flow. Subject data is collected using forms and later submitted to a central server. Data is never sent from the server back to the device. Enumerators are essentially replicating a paper-based data collection workflow, which is a unidirectional data flow.

Tuberculosis Test Results This too uses a **unidirectional** data flow. Data is transcribed from the ledgers at health centers into the forms on the PDAs carried by enumerators. At their central office they then submit data to the server.

Supply Chains Using Mobile Phones In this case the data flow is **bidirectional**. Pharmacists use a Java-based application for feature phones that sends sale and stock count information to the central server. The server processes this data and broadcasts resultant information to all users of the system via SMS. Further, the list of materials at each health center is pulled from a central server. The authors of the tool took care to degrade gracefully in the case of poor connectivity, but the flow of data in the deployment is still bidirectional.

Chimpanzee Monitoring Data collected by rangers is stored locally. It can be reviewed and revised until it is pushed to the central server. No data is ever sent from the server to the device. Further, this deployment is a replacement for paper-based data collection. All of these characteristics indicate that it employs a **unidirectional** data flow.

Aid Distribution Beneficiary information is entered on mobile devices in a digital form. Screening takes place on four mobile devices. During distribution, beneficiaries must be able to be processed on any device, not just the device that screened them. This indicates that data must be shared between devices, indicating that the data flow is **bidirectional**.

5.3.2 User Interface

Mobile deployments can be described in terms of two distinct modes defining their users' interactions: form-based and non-form-based. The distinction between these modes is motivated by two factors. First, form-based data entry is extremely common in low-resource deployments [36]. Second, a large number of tools exist that facilitate form-based data collection on mobile devices. Many are designed specifically to be leveraged by lightly technical users. This host of data-entry applications includes Google forms, CyberTracker [4], Red Cap [44], Open Data Kit (ODK) Collect [45], ODK Survey [10], Pendragon Forms, Magpi, CommCare, and many others.

One hallmark of form-based tools is that deployment architects do not normally need sophisticated control over the presentation layer of form-based data entry: presenting a piece of text indicating what data should be entered is usually sufficient. In a form-based UI, a user is making changes to the database by stepping through a series of questions, potentially with branching based on responses. Non-form-based UIs do not have as clearly defined workflow. Almost all mobile applications produced by highly technical organizations and aimed at high-resource settings (e.g. email apps, to-do lists, calendars, and chat clients) do not use a form-based workflow. This stands opposed to the applications used by organizations in low-resource settings, where workers are often employed to collect data using mobile phones.

Any workflow can be viewed through these two modes, but they have increased relevance in low-resource settings. Which of these modes is appropriate and necessary for a given deployment can define immediately the types of interactions users will have with mobile devices. Understanding and describing a deployment can be greatly simplified if deployment architects identify early if they can model their deployment's user interface using a form-based workflow. In general, the more a deployment can be forced to follow a form-based workflow, the easier it will be to manage by local organizations with tools that have a relatively low barrier to entry.

Longitudinal HIV Study The HIV study employs a predominantly **form-based** user interface. Forms were designed for screening participants and for a number of follow up visits. A lightweight non-form-based skin was designed to present a list of existing participants, but the vast majority of enumerators' time is spent completing digital forms.

Tuberculosis Test Results Here again the user interface is predominantly **form-based**. Pendragon Forms was used to create forms mimicking the data collected on paper ledgers and transcribed on PDAs. A non-form-based component was used to perform data quality checks on the server, but enumerators spent most of their time completing digital forms.

Supply Chains Using Mobile Phones Pharmacists in the supply chain deployment used a custom-built Java app for feature phones. The bulletin board aggregating results from pharmacists was written as a web page. Thus the user interface was **non-form-based**.

Chimpanzee Monitoring Although the JGI was replicating paper-based data collection, the tablet-based app did not follow a standard digital form workflow. Unlike in conventional forms, the JGI had strict requirements for the presentation layer: it must be tabular, show previously entered data to allow visual auditing, and data entry must employ stylized icons rather than text. This workflow was **non-form-based**.

Aid Distribution Both screening and distribution phases took place using digital forms, making this a **form-based** user interface.

5.3.3 Connectivity Mode

Mobile deployments can follow one of two connectivity models: connected and disconnected. A disconnected model is one where full functionality is capable without connecting to a central server. In this model, workers would be able to go into the field for periods of time and use a mobile tool without degraded quality. For example, data enumerators might leave the city for several weeks at a time collecting data about the state of a country's refrigeration infrastructure at its health centers. When they return they submit their information to their supervisor. The tasks they were expected to perform were not dependent on a reliable connection to the internet or to their superior via a telephone.

Connected operation, on the other hand, requires a connection to achieve the full functionality of an application. Mobile deployments in low-resource environments can adopt either model: organizations might have an enumerator in the field without connection or a researcher working at a large hospital with a strong WiFi connection.

Longitudinal HIV Study Many of the participants in the HIV study are expected to be contacted in the field without internet connectivity. The mobile app was designed to work entirely offline, with forms created using ODK Collect. Collect permits data entry offline, storing data locally until it is uploaded when internet connectivity becomes available. Enumerators thus were able to use the app's full functionality without an internet connection, making this a **disconnected** connectivity model.

Tuberculosis Test Results Data was entered on PDAs using Pendragon Forms. This did not require connectivity until data was uploaded at a central office. This is a **disconnected** model, as full functionality did not require a connection.

Supply Chains Using Mobile Phones The authors of the supply chain intervention took great care to ensure that their mobile devices would accommodate a **disconnected** connectivity model. After an initial download, data is persisted locally. Service degrades gracefully, defaulting to SMS data transfer if an internet connection is not available, and allowing full offline entry if neither SMS or data is available.

Chimpanzee Monitoring The mobile tool for the JGI was designed from the outset to embrace a **disconnected** connectivity model. Data can be collected entirely offline and only requires an internet connection to send data to a server.

Aid Distribution The aid distribution deployment requires a **connected** connectivity model. Distribution cannot follow screening without first aggregating all the data centrally, pairing beneficiaries with a debit card, and downloading this new information to all the devices. Further, without a connection workers cannot prevent double distribution—a beneficiary might visit two distribution stations.

5.3.4 Edit Model

Data edits within a deployment can also be characterized as non-transactional or transactional. Transactional data is data where edits are dependent on one another. The order of edits matter and are conceived of as a unit. Non-transactional data is data where the order does not matter and edits are independent of each other. For example, records of medical visits are non-transactional. Each record refers to a separate visit. Reports may be submitted out of order and remain coherent. Transactional data, meanwhile, places stricter requirements on ordering. An example is financial data, where a sequence of withdrawals and deposits must be ordered to ensure that the balance is sufficient to address subsequent requests.

Longitudinal HIV Study The HIV study treats data entry as a sequence of reports. The six month follow up is not dependent on the six week follow up. This is a **non-transactional** edit model.

Tuberculosis Test Results Again each collected data point is isolated and independent of the others, making this a **non-transactional** edit model.

Supply Chains Using Mobile Phones In this case data collected from pharmacists consists of stock counts and stock disbursements. This data is only useful if ordered. If all stock disbursements were sent two days late, the functionality of the bulletin board system would be severely impacted. Consequently this deployment requires a **transactional** edit model.

Chimpanzee Monitoring Data is collected about each time point and is independent of the others, making this a **non-transactional** edit model.

Aid Distribution A cagey beneficiary might try and cheat the system by visiting two distribution systems in order to receive double aid disbursement. This implies that ordering matters and edits are not independent of each other, making this a **transactional** edit model.

5.3.5 Server Requirements

Broadly speaking, the server requirements for a mobile deployment can be bucket-based or processed. Bucket-based servers are the simplest, acting as receptacles or sources for data. Processed servers are everything else. This distinction is purposefully broad, as the moment a server stops being bucket-based it becomes significantly more complicated. Bucket-based servers are those where all form data is submitted to a single location or pulled from a single location. Processed servers might serialize data for consumption or analyze data and perform notifications. Bucket-based server workflows can be replicated with a wide variety of tools, while processed servers require more customization and configuration.

Longitudinal HIV Study This is a classic **bucket-based** server configuration. Data from each form is sent to a table on a server. No processing is required. The forms are written using ODK Collect and the bucket-based workflow is facilitated by ODK Aggregate, which serves as a bucket for form data.

Tuberculosis Test Results This deployment uses Pendragon Forms to send data to an Oracle database. A module was added to the backend that supported validation of submitted data, highlighting errors in red. This represents a **processed** server requirement.

Supply Chains Using Mobile Phones The server in this deployment performs a number of tasks. It supports bidirectional data flow of JSON data to the Java feature phone application, it synthesizes data and presents it on a helpful bulletin board, and it sends broadcasts to registered users of critical events. This represents a high degree of customization and configuration and demonstrates what can be accomplished with a **processed** server.

Chimpanzee Monitoring Data is collected locally and sent to the server. Nothing is required of the server beyond being a receptacle for data, making it **bucket-based**.

Aid Distribution Screening and distribution data is entered using a digital form and submitted using a **bucket-based** server configuration.

5.4 *Discussion*

The power of the DUCES framework is twofold. First, it provides a schema by which to understand the requirements of a mobile deployment. Second, it provides a means by which to simplify a deployment.

5.4.1 Understanding

DUCES permits deep insight into the requirements of mobile deployments in low-resource settings. With appropriately scoped requirements, solutions can be created by leveraging existing technology. With sufficient technical knowledge and resources, custom-built solutions can be created to meet any set of requirements. Electronic medical record systems have been deployed successfully on custom technology using commercial-quality servers and custom work stations in Haiti for thousands of patients [61]. Touchscreen PCs running custom software have been used in Malawi to improve point of care treatment and provide immediate reporting to doctors in the field [29]. These are testaments to the potentially transformative power of technology, but unfortunately such technical feats are not available to a number of organizations with fewer resources.

A crucial observation is that the axes of design underpinning DUCES do not exist in isolation. A bidirectional data flow might affect server requirements, for instance, perhaps necessitating a processed configuration. The supply chain case study used bidirectional data flow to send lists of items and alerts to mobile phones. Consequently they required a processed server to synthesize data and generate alerts as well as to present a list of items to mobile phones in a specialized format (JSON). Unfortunately, however, there are no hard and fast rules when reasoning about the impacts of architectural decisions. The aid distribution case study, for example, similarly uses a bidirectional data flow but manages to use a bucket-based server. This seeming contradiction is one example of why it can be difficult for organizations to hone their intuitions surrounding technology requirements.

Why the discrepancy? In short, the supply chain case study required logic on the server that would create events and broadcast them to registered devices. It also needed to expose data using a custom format—JSON—that could be consumed by devices. The aid distribution study, meanwhile, was implemented using ODK Survey and Aggregate, which supports an out-of-the-box bucket-based server for producing and consuming data. This is difficult to recognize prima facie without extensive knowledge of the capabilities of the tools being used to implement the study. For deployment architects without a strong technical background, this will be an especially difficult conclusion to draw.

Instead, DUCES claims that the most easily satisfied configuration is **unidirectional**, **form-based**, **connected**, **non-transactional**, and **bucket-based**. If a set of requirements can be modeled using this configuration, it will be more likely to be managed successfully without outside technical resources. Deviations from this model will create additional dimensions and edge cases that will complicate the deployment and potentially require technical assistance.

For example, consider trying to support transactional data using a disconnected workflow. In the aid distribution scenario, the edit model is transactional. This is necessary as it is important that aid is not distributed twice to the same beneficiary. If errors cannot be tolerated, this requires a connected connectivity model. This is a familiar problem in distributed systems, as it is essentially an extension of the CAP theorem [35]. The CAP theorem states that a system cannot be partition tolerant, available, and provide a consistent view of the data simultaneously. In the context of the aid distribution case study, no worker would be able to become disconnected (essentially partitioning the network) while the other users are able to maintain a consistent view of the data (not double distributing) but not have to wait for all users to reconnect. This highlights a fundamental tension between a transactional edit model and disconnected workflows. The supply chain case study was able to use both transactional data and a disconnected workflow by adding processing logic to their server and, in the worst case, simply tolerating late data that was no longer actionable. This was acceptable in their deployment and was a necessary concession to support disconnected operation.

DUCES also provides insight into why mobile deployments in low-resource settings are fundamentally challenging. The simplest configuration can be at odds with the realities of the environment. It is common for many deployments to require disconnected operation, for instance, because they occur in regions without reliable data connectivity. As we have seen, this complicates the handling of transactional data but is simply unavoidable in some settings. To take another example, the chimpanzee monitoring study required a non-form-based workflow to be effective. The JGI had previously tried to encode the workflow using traditional form-based building tools without success. In the end they required a custom solution that could support their non-form-based workflow.

5.4.2 Simplification

DUCES can also be used to guide the simplification of mobile deployments. It posits that deviations from the simplest configuration of **unidirectional**, **form-based**, **connected**, **non-transactional**, and **bucket-based** will invite technical complications that may be insurmountable without the aid of a developer.

For example, bidirectional data flow is more difficult to support than unidirectional data flow. If a bidirectional data flow requirement can be loosened to a unidirectional data flow, this will have positive ramifications for the sustainability of the deployment. In many cases it is easier to alter the requirements of a deployment than to devise a sophisticated technological solution that will add complexity and hurt sustainability. We now discuss the five case studies in the context of simplification using the DUCES framework.

Longitudinal HIV Study This study has been running successfully for over three years in a hospital in Kenya. The researchers first requested a bidirectional data flow and a processed server model. Five to ten phones were to be shared between enumerators and used to screen participants and perform follow up interviews. The researchers had previously seen enumerators make errors typing subject identifiers, making it difficult to perform post-hoc analysis. They reasoned that bidirectional data flow would allow all participant records to exist on all phones. Whenever a participant was contacted for follow up, enumerators would not have to re-type the identifier, reducing the likelihood of errors.

However, supporting bidirectional data flow would complicate requirements. First, the server would have to support presenting captured data in a machine-consumable way, similar to how the supply chain example provided JSON. Second, it might require authentication and access control to prevent the collected HIV data from being visible to non-study devices. Third, enumerators would have required a stable internet connection at headquarters before leaving for the field. If a connection issue interrupted the bidirectional data flow, the flow might be interrupted.

Instead, the study designers were able to refine their study procedures to work within the confines of a unidirectional data flow. Enumerators were made responsible for the same cohort of patients and assigned specific devices rather than a shared device. This greatly increased the likelihood that a participant would already be present on the device without requiring bidirectional data flow. However, participants might still be seen for follow up interviews on devices that were not used to screen them. This might occur if the screening device was lost or stolen or if they were visited by a new enumerator for logistical reasons. To accommodate this eventuality, the follow up workflow was modified to allow entering an existing patient identifier. This was not completely in-line with the original requests of the researchers, as they requested that the identifier not be entered manually more than once, but with the advent of individually assigned devices the likelihood of a manually entered identifier was less common and was deemed acceptable.

A processed server was desired to calculate when participants were due for follow up visits. This information would be generated each morning and provided to study coordinators. Automating this task would not be complicated for a computer scientist, but the smooth operation of the study would depend on this task functioning without interruption. This might prove difficult without a technical staff capable of supporting the server. Instead, the researchers directed study staff to manually review the data on the bucket-based server and generate a list of follow up participants each day by hand. This might seem less than elegant to a computer scientist, but it is much more sustainable with local talent.

In this way a bidirectional, processed workflow was transformed and simplified to use a unidirectional, bucket-based workflow. These simplifications are a large part of the reason that the study has remained in successful operation with limited involvement from outside technical staff for over two years.

Tuberculosis Test Results The authors were not involved in this deployment, so DUCES will be used to describe how the deployment might have been further simplified from its current incarnation rather than how it was simplified in practice. The configuration was almost an ideal DUCES configuration, being unidirectional, form-based, disconnected, and non-transactional. The server, however, was processed, performing validation logic and displaying it as a web page. This required adding a module to an Oracle backend managed by an NGO [5].

Although the authors do not state it, this likely required a developer with the technical ability to create a web page and encode validation logic. This processed server requirement may have been able to yield to an unprocessed server if server-side validation was not automated. Instead, data could have been exported to a format like comma-separated values (CSV) that can be consumed by a number of programs. At that point it could be manually validated by a staff member, or validation logic could be encoded in a Microsoft Excel worksheet rather than a web page. This would slow the cycle of validation but would not require web programming skills. Alternatively, validation logic is supported by a number of form-based data entry tools. The researchers could instead have performed validation upon data entry rather than during server auditing. Both of these solutions would yield a near optimal configuration under the DUCES framework.

Supply Chains Using Mobile Phones This deployment serves as a testament to the rich functionality that can be achieved using custom solutions. A custom Java application for feature phones communicated with a custom processed server capable of performing analysis and broadcasting alerts to users. Here again the authors were not involved with the deployment, so application of the DUCES framework will be aim to demonstrate how an organization with less technical resources might try to replicate the success of this workflow.

First, the non-form-based workflow for feature phones could be replaced by a form-based data entry tool for smart phones. As discussed in Section 5.3.2, a number of form-based data entry tools have been designed to support use by non-programmers.

The processed server model is crucial to this deployment. The authors of the study argue that consumers of the information submitted by the mobile devices are too busy to synthesize the reports without automation. This domain knowledge suggests that simply converting to a bucket-based server model is inappropriate. The authors also note explicitly that their edit model is transactional and that their mobile application functions offline. It is informative to look at how they circumvent the requirement put forth in Section 5.4.1 that transactional workflows require connectivity.

The answer is twofold. First, they apply server-side processing to deduplicate and process errors that are created as a result of network errors. Second, although their data is transactional, they are able to tolerate errors. In terms of the CAP theorem, they are able to tolerate a loss in consistency as long as the system remains available during periods of no data connectivity. The ramifications for the deployment are that events might not be shown on their web-based bulletin board in real-time. A stock out might be reported late, but this is likely uncommon and is thus deemed acceptable.

Chimpanzee Monitoring The chimpanzee monitoring case study attains a near-optimal DUCES configuration. It fails by being non-form-based and disconnected. In reality the only simplification that might be afforded by a connected model would be that a wider variety of tools could be used to implement the framework. This would thus accommodate a wider range of user interface components and backends, including potentially a web-based application. Practical implications of this change are low due to the fact that the data is non-transactional and thus ordering is not significant.

Arriving at this configuration was straight-forward, as the JGI was seeking to replace an existing paper workflow. It is important to note that the non-form-based workflow necessitated the involvement of the assistance of developers, which is an added technical burden. The JGI has extensive experience creating form-based workflows, but the creation of a non-form-based workflow requires an additional skill set. In this case the DUCES framework did not simplify the deployment, but it did clearly delineate where external resources would be required.

Aid Distribution The aid distribution case study was bidirectional, form-based, connected, transactional, and bucket-based. In the HIV case study, the bidirectional data requirement was able to be eliminated by having participants ideally interact with only a single device, obviating the need to share data between multiple devices. This was not possible in the aid distribution case study, as the nature of the distribution environment required that beneficiaries not be confined to an individual device. With this requirement, tools immediately had to be chosen that could support bidirectional data flow. This eliminated some possibilities like Pendragon Forms, Magpi, and ODK Collect, which support only unidirectional data flows.

A connected connectivity model was required to accommodate the transactional nature of the data. In this case, if connectivity was lost, workers might see an inconsistent view of the data. In other words, a beneficiary may have received aid from one disconnected distribution station and then received aid from a second station that was not aware aid had already been distributed. To prevent this, the system required a connected model. This would in turn require that whatever tools were used to implement the deployment support an online, connected workflow. However, this requirement was able to be circumvented by the real-world details of the aid distribution.

In this case the aid itself was a debit card that had been uniquely assigned to each beneficiary. Once distributed, it could not be distributed again, preventing double distribution. This relaxes the requirement slightly, although at the cost of masking errors during distribution. Coordinators would know a card was missing, but they would not know if it had already been given to the correct recipient or had simply been lost or misplaced. Further, this approach would not accommodate distribution of goods that were not uniquely paired to beneficiaries.

5.5 Conclusion

Mobile devices are increasingly integrated into the workflows of organizations working in low-resource settings. We have presented the DUCES framework as a means to elucidate the requirements of data-focused mobile deployments. We have described five case studies with varying aims and requirements and discussed how they can be evaluated under the DUCES framework, permitting both a meaningful understanding of requirements and guiding simplifying assumptions. Using this framework, deployment architects can start to identify what requirements will add significant complexity. This in turn facilitates the selection of technologies. If a deployment requires bidirectional data flow and that requirement cannot be loosened, technologies that do not support bidirectional data flow can be discounted immediately.

It is important to note that the DUCES framework is not simply an attempt at formalizing a series of known tradeoffs in system design. For example, encrypting data can result in decreased usability while increasing data security. DUCES instead provides a mental scaffold by which a single set of requirements can be described and satisfied in a number of different ways. A processed server might require a custom-built solution with a custom database and a suite of scripts running every night. It instead might be transformed into a bucket-based server by having an administrator copy relevant rows between data sinks and data sources. This would seem an ugly solution to a computer scientist that prefers to automate all tasks. However, it would elegantly allow a lightly skilled deployment architect to compose simple tools in a powerful way that meets the needs of their deployment while remaining comfortably within their skill set.

Some things will always remain difficult in mobile deployments conducted by organizations and groups with limited resources. Transferring data between backends is a prime example. Inputting data collected into another system will always be difficult. No choice of tools will completely allay this difficulty unless a programmer has already taken the time to create a method by which data can be exported in a form consumable by the tool in question. Organizations are better off recognizing that this will require technical expertise than they are limiting their technological solutions to one that will be compatible with their current target repository out of the box.

The DUCES framework provides a useful set of considerations for architects of mobile deployments and data-based workflows. The framework can be applied to all systems, but it is most effective when considered by deployment architects operating in low-resource settings. In these environments DUCES can be used to re-imagine requirements in ways that will make mobile workflows easier to deploy and maintain. The DUCES framework is a valuable tool that organizations in low-resource settings can use to characterize and simplify their data-focused mobile deployments.

Chapter 6
CONCLUSION

This dissertation has shown that technology can be built for the developing world using existing infrastructure. It has presented two tools that rely on devices already present in the technological landscape to enhance workflows. With tools like these available, the question then becomes how to deploy them effectively. The DUCES model for discovering and reducing complexity in mobile workflows addresses this by providing a scaffold to understand technological requirements. An approach that focuses on existing workflows rather than creating new ones may be ill-advised if the requirements of developing world deployments are fundamentally different than those in the developed world. However, the security study of Open Data Kit (ODK) deployments shows that, at least in some ways, many of these challenges are shared in developed settings. Taken together, these projects have demonstrated that targeting existing infrastructure is a powerful approach to Information and Communication Technologies for Development (ICTD).

6.1 Benefits of Existing Infrastructure

Targeting existing infrastructure has several ancillary benefits. While my work did not characterize these benefits explicitly, I believe they are relevant and bear repeating.

First, targeting existing infrastructure leads to greater deployability. Projects built for what already exists, or for what can exist with modest modifications, are more field-ready than projects that require specialized hardware to be deployed. This in turn can lead to faster adoption and more immediate impact.

Second, this approach is more likely than not align with autonomous local processes. This is advantageous because it is more likely to augment rather than dominate local practices. Existing infrastructure is by its nature infrastructure that someone in the field has already deemed important. Rather than convince a rural community that they need mobile phones, targeting existing technology has an opportunity to be in line with trends underway on the ground. This is not always the case, and infrastructure could still be the product of a flawed government plan to force laptops or cell towers onto people that do not want them. As a general rule, however, I believe that targeting existing infrastructure is more likely than not to align with local interests.

Finally, targeting existing infrastructure minimizes guessing on the part of implementers. The value of local context is well known in ICTD, and participatory design has emerged as a prominent force in the field [6]. Although a valuable approach, it strikes me that the logical conclusion of these hyper-local arguments is that it would be better for implementers in the developed world to mind their own business. However, it is a resource-rich individual that can afford to leave their home in Seattle, embed in a local village for eight months, and design a system with the input of the village council. Of course this is a valuable approach, but it should not be the only valid approach. Rather than recognize that they cannot fully understand another's context and thus not bother trying, implementers might treat existing infrastructure as a piece of the picture on the ground and build for it. Not every project will be useful, but innovations and insight can occur that then can be applied by others, including locals themselves, that leverage these discoveries.

6.2 Deployments in ICTD Research

Siskin discusses an implementation and prototype, evaluating the technology in a laboratory setting. DUCES and the discussion of security focus on the experience of users in the field. Tables takes a hybrid approach, including a presentation of a technology as well as deployment experiences. Deployment experiences are a key component of many Information and Communication Technologies for Development (ICTD) papers. This is a strength when it leads to insight into an issue that cannot be uncovered without going to the field. However, it is easy for a project to be legitimized as a valuable research contribution by including a deployment, even if the deployment was not crucial to the underlying insight. In turn, researchers end up devoting time to deployments, frequently sacrificing technical insights or foregoing ideas for which they do not have deployment partners.

I would like to use this space to discuss the role deployments play in ICTD. These arguments are based on experiences with my own work, where I have seen both the value and the facade of deployments in ICTD research projects. Ultimately I believe that papers with and without deployment components should be considered separately, rather than compared to one another. As I argue below, this will lead to faster iteration, a wider array of projects in ICTD, and be better for the field.

6.2.1 Misplaced Emphasis on Deployments

ICTD is an unusual field. It is a bit of a catch-all, and a paper is as likely to be by a social scientist as by a computer scientist. Being multidisciplinary invites different perspectives and can be a great boon to a field. It can also lead to muddled purpose and expectations.

One way this manifests itself is by putting a premium on real-world deployments. Proponents of deployments make the case that their value comes from the fact that ICTD is motivated by the desire to impact development. Without seeing the effects of the technology in the real world, invaluable context will inevitably be lost, limiting the value of the work. A side effect of this line of thinking is that deployments themselves begin to take on value, and deployments are done for their own sake rather than to appreciate context or validate an approach. Even if the technical contribution of a project is minimal, if it was a successful deployment—e.g. prominent partner organizations, a large number of users, or as a business—the deployment box is checked and the work has a decent chance of being published.

On its face this is not necessarily a bad thing. There is absolutely a place for deployment-based research projects. Some insights, like those that are the focus of Human Computer Interaction (HCI) studies, rightly depend on context. In other cases, however, the drive to deploy can create situations that are not ideal. In the worst case, it gives rise to a perfect mismatch of expectations between implementers and field partners. For example, in my own work I have frequently partnered with global health researchers. These practitioners are typically not technology experts, but their work increasingly depends on technology deployments. As a computer science researcher in ICTD, I feel the obligation to deploy my tools in the field in order to publish. On its surface, this might seem like a match made in heaven—I need field partners, while the researchers need technical expertise.

In my experience, however, our requirements are exactly at odds with one another. The researchers need something reliable and foolproof, while what I have to offer is a prototype. Prototypes by their very nature are not reliable and foolproof. As a researcher I should not be obligated to productize every project, package it and ship it. I believe that as a research project, a prototype can have value even if it never sees the field.

What is more, say that a paper contains a section with a field deployment. Does that say anything about the maturity of the project or the validity of the idea? I argue that it does not. It shows that the researcher chose a good field partner, or that the researcher is a capable marketer, but it says nothing about the value of the research itself. A good project might have a good deployment, but the presence of a deployment component is a very poor indicator of a good project. This is similar to the problem referred to as "pilotitis" [81]. Paraphrasing the sociologist Peter Henry Rossi, "there is a big difference between running a program on a small scale with highly skilled and very devoted personnel and running a program with the lesser skilled and less devoted personnel that [a larger deployment] will have at its disposal" [73]. The same is true of ICTD research. People treat deployments as if they are indicators of maturity or of the success of future, larger efforts. In my experience, this is not the case. Scaling up invites a host of technical and personnel problems that cannot be modeled in a deployment.

Looking at it another way, what does it mean for a deployment to be a failure? Something can be gleaned even from the most beleaguered deployments, but that does not mean that the deployment adds value. If a project has a promising prototype and conducts a field deployment, that paper has a good chance of being published, regardless of the findings of the deployment. If partners fall through, a paragraph is added speaking to the complexities of navigating unfamiliar power structures. If users struggle to understand a picture-based user interface, the authors opine about the value of local languages that are under-served by font providers and mobile phones, or education systems that do not lead to literacy. These results can no longer be claimed as findings. As a discipline we are well aware of these issues, and they are familiar to anyone who has spent time in the field.

6.2.2 Recapturing the Value of Deployments

None of this is to say that deployments are worthless. Certainly the context that arises from a field deployment can yield crucial insight, even beyond the perfunctory observations present in every field study—local context is important, the social element is a crucial part of a technology, sustainability is challenging, etc. However, I think that ICTD papers should be required to explicitly indicate whether or not they include a deployment component. Those that do should not be compared to those that do not. Anyone that has been involved with a deployment recognizes the herculean amount of work suggested by a six month field study with 200 users that began with user-centered design to identify and address a problem. However, that work should not be compared to a paper that explores a technical concept but that does not rise to the level of deployment. Not every research idea is appropriate to be deployed, but that does not mean it is not worthy of publication.

I believe that ICTD publications would become more impactful, more diverse, and invite more iteration, if deployments were not expected. Deployment papers should be considered separately from those evaluated solely in a laboratory setting. Inclusion of perfunctory findings would be reduced, the rate of iteration would increase, and more interesting work would be possible.

A natural response to this criticism is to point at other areas of computer science where deployment papers can be considered a gold standard. For example, Google's publications on Map Reduce, Big Table, and the Chubby locking system have had a profound impact on both industry and research [26, 13, 11]. Papers on Amazon's DynamoDB and Facebook's Cassandra have served as validation of techniques for making scalable database systems, and those techniques are now canonical [27, 58]. Such papers would not have had such an impact were it not for the deployment components. Deployment in these cases took the idea and proved that it works in practice.

Proponents of deployments in ICTD believe in the power of validation. The problem with this line of reasoning is that an academic deployment is fundamentally different from a production-scale integration of a tool into the workflows of some of the most technically sophisticated companies in the world. Knowing that an approach is sufficient for Google's needs means more than knowing that I was able to deploy a mobile application for three weeks with a group of friendly researchers in the field. The similarities exist in name only.

By not emphasizing the value of deployments, it is possible that projects will lose touch with the communities they aim to serve, or that it will become impossible to judge the maturity of an idea or an implementation. These are valid concerns, but both can be addressed.

First, consider the issue of losing touch with context on the ground. Local context is impossible to replicate in a lab. However, my proposed solution does not eliminate deployment papers. Instead it tries to separate deployment and non-deployment papers as different classes of work. A deployment paper has merit even if the technical idea behind it is not novel. By considering both classes of work alongside each other, venues and program committees are limiting the amount of technical innovation that can occur. This is a loss for the field and a loss for development, as it prevents researchers from publishing discoveries that do not lend themselves to field deployments.

Second, without deployments how will technical maturity be evaluated? My primary response to this argument is that in reality a field deployment proves next to nothing about technical maturity. I am aware of numerous ICTD academic projects that worked only in the hands of their implementers. Outside sources would have had an extremely difficult time cobbling together a similar deployment. My own Tables deployments would be difficult to replicate by someone that did not write the code, as I did not create documentation for every step of the process and the project has continued to evolve. This does not undermine the value of the work. More convincing than the field deployment is the fact that all of the code I used to generate the project is free and open source, posted and available online. Computer scientists are fortunate that their work can be published so easily. Biologists are not able to make cell lines or reagents online and replicable by any that wish to see their work. In lieu of a deployment, reviewers should expect the code, experimental set up, and data to be available online. As a community we should be able to expect projects to build and experiments to be rigorously documented.

I believe this change would be a boon for the field. Innovations would not be limited by the need for deployments, and technical rigor would increase. Researchers that do not benefit from strong partners would not be relegated to secondary publication venues. Projects that are both technically rigorous and feature deployments would retain their position at the forefront of the field. In the long run, separating these two classes of work would enrich the research conducted within the discipline of ICTD.

6.3 Final Remarks

Technology can be designed for the developing world by targeting existing infrastructure. I have shown how this can be accomplished using new technologies and by understanding the technological requirements of workflows. Additionally, by interacting with people using technology in the developing world, I have shown that, at least in some ways, problems in the developing world do not present a fundamentally new class of problems to those in the developed world. Targeting existing infrastructure provides researchers and implementers with an approach that respects local context and acknowledges realities on the ground. Leveraging existing infrastructure is a valuable way forward for ICTD that will lead to deployable innovations while continuing to help those who need it most.

6.4 Acknowledgments

This work was supported by the University of Washington, the National Science Foundation, USAID, and Google. It would not have been possible without the cooperation and support of numerous field partners, and I would like to extend my sincere thanks to the people I have worked with. To name a few: The Jane Goodall Institute; Drs Joe Zunt and Patty Garcia at the Kuskaya Fellowship; Drs Magaly Blas and Isaac Alva and those at Mamás del Río; The Hope Study; and Lorenzo Violante and Alberto Cabrera at the International Federation of Red Cross and Red Crescent Societies.

Thanks also to my ICTD labmates at the University of Washington for enduring my pontifications for the past six years. No doubt my thoughts in this last chapter are all too familiar to many of them. Our developers Clarice Larson, Mitch Sundt, and Jeff Beorse have fixed many of my bugs and have patiently explained many things to me. May they escape the technical debt I've left them.

I would never have finished without Richard Anderson and Matt Welsh advising me for the past two years. Thank you both.

And most of all I would like to thank Gaetano Borriello, my first ICTD advisor. There is no doubt in my mind that I would have left the department early, which I would have regretted, had Gaetano not made the perhaps ill-advised decision to take me on as a student. Four years later, my biggest thanks belong to him.

BIBLIOGRAPHY

[1] Fundación Sergio Paiz Andrade. “Evaluation Report: Assessing the use of technology and Khan Academy to improve educational outcomes in Sacatepéquez, Guatemala”. In: (2016).

[2] Yahel Ben-David et al. “Computing Security in the Developing World: A Case for Multidisciplinary Research”. In: *NSDR ’11*. Bethesda, Maryland, USA: ACM, 2011, pp. 39–44. ISBN: 978-1-4503-0739-0. DOI: `10.1145/1999927.1999939`. URL: `http://doi.acm.org/10.1145/1999927.1999939`.

[3] Prasanta Bhattacharya and William Thies. “Computer Viruses in Urban Indian Telecenters: Characterizing an Unsolved Problem”. In: *NSDR ’11*. Bethesda, Maryland, USA: ACM, 2011, pp. 45–50. ISBN: 978-1-4503-0739-0. DOI: `10.1145/1999927.1999940`. URL: `http://doi.acm.org/10.1145/1999927.1999940`.

[4] Edwin H Blake. “Extended abstract a field computer for animal trackers”. In: *CHI 02 extended abstracts on Human factors in computing systems CHI 02* (2002), p. 532. DOI: `10.1145/506461.506466`. URL: `http://portal.acm.org/citation.cfm?doid=506443.506466`.

[5] Joaquın a. Blaya et al. “Personal digital assistants to collect tuberculosis bacteriology data in Peru reduce delays, errors, and workload, and are acceptable to users: cluster randomized controlled trial”. In: *International Journal of Infectious Diseases* 13.3 (2009), pp. 410–418. ISSN: 12019712. DOI: `10.1016/j.ijid.2008.09.015`.

[6] Paul Braund and Anke Schwittay. “The missing piece: Human-driven design and research in ICT and development”. In: *Information and Communication Technologies and Development, 2006. ICTD’06. International Conference on*. IEEE. 2006, pp. 2–10.

[7] Andrew Brown and Jeffrey S. Chase. "Trusted Platform-as-a-service: A Foundation for Trustworthy Cloud-hosted Applications". In: *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. CCSW '11. Chicago, Illinois, USA: ACM, 2011, pp. 15–20. ISBN: 978-1-4503-1004-8. DOI: `10.1145/2046660.2046665`. URL: `http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2046660.2046665`.

[8] Waylon Brunette et al. "ODK Tables: Building Easily Customizable Information Applications on Android Devices". In: *Proceedings of the 3rd ACM Symposium on Computing for Development*. ACM DEV '13. Bangalore, India: ACM, 2013, 12:1–12:10. ISBN: 978-1-4503-1856-3. DOI: `10.1145/2442882.2442898`. URL: `http://doi.acm.org/10.1145/2442882.2442898`.

[9] Waylon Brunette et al. "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*. HotMobile '13. Jekyll Island, Georgia: ACM, 2013, 10:1–10:6. ISBN: 978-1-4503-1421-3. DOI: `10.1145/2444776.2444790`. URL: `http://doi.acm.org/10.1145/2444776.2444790`.

[10] Waylon Brunette et al. "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 - HotMobile '13* (2013), p. 6. DOI: `10.1145/2444776.2444790`.

[11] Mike Burrows. "The Chubby lock service for loosely-coupled distributed systems". In: *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, pp. 335–350.

[12] Kelly Caine, Selma Sabanovic, and Mary Carter. "The effect of monitoring by cameras and robots on the privacy enhancing behaviors of older adults." In: *HRI*. 2012.

[13] Fay Chang et al. "Bigtable: A distributed storage system for structured data". In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), p. 4.

[14] Rohit Chaudhri et al. "FoneAstra: enabling remote monitoring of vaccine cold-chains using commodity mobile phones". In: *Proceedings of the First ACM Symposium on Computing for Development*. ACM. 2010, p. 14.

[15] Eric Y. Chen et al. "OAuth Demystified for Mobile Application Developers". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 892–903. ISBN: 978-1-4503-2957-6. DOI: `10.1145/2660267.2660323`. URL: `http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2660267.2660323`.

[16] Jay Chen et al. "Analyzing and Accelerating Web Access in a School in Peri-urban India". In: *Proceedings of the 20th International Conference Companion on World Wide Web*. WWW '11. Hyderabad, India: ACM, 2011, pp. 443–452. ISBN: 978-1-4503-0637-9. DOI: `10.1145/1963192.1963358`. URL: `http://doi.acm.org/10.1145/1963192.1963358`.

[17] Kuang Chen et al. "Shreddr: pipelined paper digitization for low-resource organizations". In: *Proceedings of the 2nd ACM Symposium on Computing for Development - ACM DEV '12* (2012), p. 1. DOI: `10.1145/2160601.2160605`. URL: `http://dl.acm.org/citation.cfm?id=2160601.2160605`.

[18] Stuart Cheshire and Marc Krochmal. *DNS-based service discovery*. Tech. rep. 2013.

[19] Stuart Cheshire and Marc Krochmal. *Multicast dns*. Tech. rep. 2013.

[20] Byung-Gon Chun et al. "Mobius: Unified Messaging and Data Serving for Mobile Apps". In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys '12. Low Wood Bay, Lake District, UK: ACM, 2012, pp. 141–154. ISBN: 978-1-4503-1301-8. DOI: `10.1145/2307636.2307650`. URL: `http://doi.acm.org/10.1145/2307636.2307650`.

[21] Jeremy Clark and Urs Hengartner. "Panic Passwords : Authenticating under Duress". In: *HotSec* (2008), pp. 6–16. URL: `http://dl.acm.org/citation.cfm?id=1496671`.

1496679$%5Cbackslash$nhttp://users.encs.concordia.ca/%7B~%7Dclark/papers/2008%7B%5C_%7Dhotsec.pdf.

[22] Camille Cobb et al. "Computer Security for Data Collection Technologies". In: *Proceedings of the Eighth International Conference on Information and Communication Technologies and Development.* ICTD '16. Ann Arbor, MI, USA: ACM, 2016, 2:1–2:11. ISBN: 978-1-4503-4306-0. DOI: 10.1145/2909609.2909660. URL: http://doi.acm.org/10.1145/2909609.2909660.

[23] Henry Corrigan-Gibbs and Jay Chen. "FlashPatch: Spreading Software Updates over Flash Drives in Under-connected Regions". In: *ACM DEV '14.* San Jose, California, USA: ACM, 2014, pp. 1–10. ISBN: 978-1-4503-2936-1. DOI: 10.1145/2674377.2674384. URL: http://doi.acm.org/10.1145/2674377.2674384.

[24] *Critical Links C3.* http://c3.critical-links.com/. 2017.

[25] Alexei Czeskis et al. "Parenting from the Pocket: Value Tensions and Technical Directions for Secure and Private Parent-teen Mobile Safety". In: *SOUPS.* 2010.

[26] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492.

[27] Giuseppe DeCandia et al. "Dynamo: amazon's highly available key-value store". In: *ACM SIGOPS operating systems review* 41.6 (2007), pp. 205–220.

[28] *Djinni, https://github.com/dropbox/djinni.* Dec. 2014. URL: https://github.com/dropbox/djinni.

[29] Gerald P. Douglas et al. "Using Touchscreen electronic medical record systems to support and monitor national scale-up of antiretroviral therapy in Malawi". In: *PLoS Medicine* 7.8 (2010). ISSN: 15491277. DOI: 10.1371/journal.pmed.1000319.

[30] *Dropbox, https://www.dropbox.com.* Dec. 2014. URL: https://www.dropbox.com.

[31] *Dropbox Tech Talk: High-Speed and High Performance Mobile Engineering*. Tech talk at the University of Washington describing the motivations and architecture of Djinni. Oct. 2014.

[32] Markus Dürmuth. "Useful Password Hashing: How to Waste Computing Cycles with Style". In: *Proceedings of the 2013 Workshop on New Security Paradigms Workshop*. NSPW '13. Banff, Alberta, Canada: ACM, 2013, pp. 31–40. ISBN: 978-1-4503-2582-0. DOI: `10.1145/2535813.2535817`. URL: `http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2535813.2535817`.

[33] *eGranary*. `https://www.widernet.org/eGranary/`. 2017.

[34] Li Fan et al. "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol". In: *IEEE/ACM Trans. Netw.* 8.3 (June 2000), pp. 281–293. ISSN: 1063-6692. DOI: `10.1109/90.851975`. URL: `http://dx.doi.org/10.1109/90.851975`.

[35] Armando Fox and Eric Brewer. "Harvest, yield, and scalable tolerant systems". In: *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems* (1999). ISSN: 15301621. DOI: `10.1109/HOTOS.1999.798396`.

[36] Hamish SF Fraser et al. "Information Systems for Patient Follow-Up and Chronic Management of HIV and Tuberculosis: A Life-Saving Technology in Resource-Poor Areas". In: *Journal of medical Internet research* 9.4 (2007).

[37] Nicci C. L. Gafinowitz. "Digital Library Appropriation in the Context of SubSaharan Countries: The Case of eGranary Digital Library Implementation". In: *Proceedings of the 7th Annual Symposium on Computing for Development*. ACM DEV '16. Nairobi, Kenya: ACM, 2016, 29:1–29:4. ISBN: 978-1-4503-4649-8. DOI: `10.1145/3001913.3006638`. URL: `http://doi.acm.org/10.1145/3001913.3006638`.

[38] Samson Gejibo et al. "Secure data storage for mobile data collection systems". In: *International Conference on Management of Emergent Digital EcoSystems* (2012), p. 131.

DOI: 10.1145/2457276.2457300. URL: http://dl.acm.org/citation.cfm?id=2457276.2457300.

[39] S Gejibo et al. "Secure cloud storage for remote mobile data collection". In: *ACM International Conference Proceeding Series* (2013). DOI: 10.1145/2513534.2513538. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-84884646951%7B%5C&%7DpartnerID=40%7B%5C&%7Dmd5=3377006f3dc636a7ee1cfe448f860527.

[40] Shimin Guo et al. "Very low-cost internet access using KioskNet". In: *ACM SIGCOMM Computer Communication Review* 37.5 (2007), pp. 95–100.

[41] Abhishek Gupta et al. "Simplifying and Improving Mobile Based Data Collection". In: *Proceedings of the Sixth International Conference on Information and Communications Technologies and Development: Notes - Volume 2*. ICTD '13. Cape Town, South Africa: ACM, 2013, pp. 45–48. ISBN: 978-1-4503-1907-2. DOI: 10.1145/2517899.2517929. URL: http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2517899.2517929.

[42] Abhishek Gupta et al. "Simplifying and improving mobile based data collection". In: *Proceedings of the Sixth International Conference on Information and Communications Technologies and Development Notes - ICTD '13 - volume 2* (2013), pp. 45–48. DOI: 10.1145/2517899.2517929. URL: http://dl.acm.org/citation.cfm?doid=2517899.2517929.

[43] Shuai Hao et al. "Building a Delay-Tolerant Cloud for Mobile Data". In: *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management - Volume 01*. MDM '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 293–300. ISBN: 978-0-7695-4973-6. DOI: 10.1109/MDM.2013.43. URL: http://dx.doi.org/10.1109/MDM.2013.43.

[44] Paul Harris et al. "Research electronic data capture (REDCap)-A metadata-driven methodology and workflow process for providing translational research informatics support". In: *Journal of Biomedical Informatics* 42.2 (2009), pp. 377–381. ISSN: 15320464.

DOI: 10.1016/j.jbi.2008.08.010. URL: http://dx.doi.org/10.1016/j.jbi.2008.08.010.

[45] Carl Hartung et al. "Open Data Kit: Tools to Build Information Services for Developing Regions". In: *Proceedings of the International Conference on Information and Communication Technologies and Development* (2010). DOI: 10.1145/2369220.2369236.

[46] Shaddi Hasan et al. "The Challenges of Scaling WISPs". In: *Proceedings of the 2015 Annual Symposium on Computing for Development*. DEV '15. London, United Kingdom: ACM, 2015, pp. 3–11. ISBN: 978-1-4503-3490-7. DOI: 10.1145/2830629.2830637. URL: http://doi.acm.org/10.1145/2830629.2830637.

[47] Kurtis Heimerl et al. "Expanding Rural Cellular Networks with Virtual Coverage." In: *NSDI*. 2013, pp. 283–296.

[48] YoonSung Hong, Hilary K. Worden, and Gaetano Borriello. "ODK Tables: Data Organization and Information Services on a Smartphone". In: *Proceedings of the 5th ACM Workshop on Networked Systems for Developing Regions*. NSDR '11. Bethesda, Maryland, USA: ACM, 2011, pp. 33–38. ISBN: 978-1-4503-0739-0. DOI: 10.1145/1999927.1999937. URL: http://doi.acm.org/10.1145/1999927.1999937.

[49] *HTTP Archive*. http://httparchive.org/. 2017.

[50] Hibah Hussain. "Dialing Down Risks". In: (2013), p. 22. URL: http://newamerica.net/publications/policy/dialing%5C_down%5C_risks%5C_mobile%5C_privacy%5C_and%5C_information%5C_security%5C_in%5C_global%5C_development.

[51] *iCloud, https://www.icloud.com/*. Dec. 2014. URL: https://www.icloud.com/.

[52] *iHub*. https://ihub.co.ke/. 2017.

[53] *Intel CAP*. https://www-ssl.intel.com/content/www/us/en/education/products/content-access-point.html. 2017.

[54] Sibren Isaacman and Margaret Martonosi. “The C-LINK system for collaborative web usage: A real-world deployment in rural Nicaragua”. In: *Workshop on Networked Systems for Developing Regions*. 2009.

[55] Sushant Jain, Kevin Fall, and Rabin Patra. *Routing in a delay tolerant network*. Vol. 34. 4. ACM, 2004.

[56] *Khan Academy Lite*. `https://learningequality.org/ka-lite/`. 2017.

[57] Pamela A Kolopack, Janet A Parsons, and James V Lavery. “What makes community engagement effective?: lessons from the Eliminate Dengue Program in Queensland Australia”. In: *PLoS neglected tropical diseases* 9.4 (2015), e0003713.

[58] Avinash Lakshman and Prashant Malik. “Cassandra: a decentralized structured storage system”. In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40.

[59] Stevens Le Blonde et al. “A Look at Targeted Attacks through the Lense of an NGO”. In: *USENIX Security*. San Diego, CA US, 2014.

[60] Andrew Y Lindell. “Comparison-based key exchange and the security of the numeric comparison mode in Bluetooth v2. 1”. In: *Cryptographers' Track at the RSA Conference*. Springer. 2009, pp. 66–83.

[61] William B. Lober, Stephen Wagner, and Christina Quiles. “Development and implementation of a loosely coupled, multi-site, networked and replicated electronic medical record in Haiti”. In: *ACM SIGOPS Operating Systems Review* 43.4 (2010), p. 79. ISSN: 01635980. DOI: `10.1145/1713254.1713272`.

[62] Radhika Malpani, Jay Lorch, and David Berger. “Making World Wide Web caching servers cooperate”. In: *Proceedings of the Fourth International World Wide Web Conference*. Boston, MA, Dec. 1995, pp. 107–117. URL: `https://www.microsoft.com/en-us/research/publication/making-world-wide-web-caching-servers-cooperate/`.

[63] F Mancini and Ka Mughal. "Adding security to mobile data collection". In: *e-Health Networking Applications and Services* (2011), pp. 498–501. DOI: `0.1109/HEALTH.2011.6026793`. URL: `http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=6026793`.

[64] Dzmitry Marushka and Maria Ablameyko. "Belarus and the Baltic States: Comparison Study on e-Government Development Results". In: *Proceedings of the 7th International Conference on Theory and Practice of Electronic Governance*. ICEGOV '13. Seoul, Republic of Korea: ACM, 2013, pp. 382–383. ISBN: 978-1-4503-2456-4. DOI: `10.1145/2591888.2591970`. URL: `http://doi.acm.org/10.1145/2591888.2591970`.

[65] Susan E. McGregor et al. "Investigating the Computer Security Practices and Needs of Journalists". In: *USENIX Security*. 2015.

[66] *Open Data Kit, https://opendatakit.org*. Dec. 2014. URL: `https://opendatakit.org`.

[67] Ismael Peña-López et al. "World development report 2016: Digital dividends". In: (2016).

[68] *PhoneGap, http://phonegap.com/*. Dec. 2014. URL: `http://phonegap.com/`.

[69] *RACHEL Offline*. `https://racheloffline.org/`. 2017.

[70] Arun Ramanujapuram and Anup Akkihal. "Improving Performance of Rural Supply Chains Using Mobile Phones: Reducing Information Asymmetry to Improve Stock Availability in Low-resource Environments". In: *ACM DEV* (2014), pp. 11–19.

[71] Bradley Reaves et al. "Mo(bile) Money , Mo(bile) Problems: Analysis of Branchless Banking Applications in the Developing World". In: *USENIX Security* (2015).

[72] Nabeel Abdur Rehman et al. "Fine-grained dengue forecasting using telephone triage services". In: *Science advances* 2.7 (2016), e1501215.

[73] Peter Rossi. "The iron law of evaluation and other metallic rules". In: *Research in social problems and public policy* 4.1987 (1987), pp. 3–20.

[74] Christopher J Seebregts et al. “The OpenMRS implementers network”. In: *International journal of medical informatics* 78.11 (2009), pp. 711–720.

[75] Ion Stoica et al. “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM ’01. San Diego, California, USA: ACM, 2001, pp. 149–160. ISBN: 1-58113-411-8. DOI: `10.1145/383059.383071`. URL: `http://doi.acm.org/10.1145/383059.383071`.

[76] Samuel R. Sudar and Richard Anderson. “DUCES: A Framework for Characterizing and Simplifying Mobile Deployments in Low-Resource Settings”. In: *Proceedings of the 2015 Annual Symposium on Computing for Development*. DEV ’15. London, United Kingdom: ACM, 2015, pp. 23–30. ISBN: 978-1-4503-3490-7. DOI: `10.1145/2830629.2830653`. URL: `http://doi.acm.org/10.1145/2830629.2830653`.

[77] Samuel Sudar, Waylon Brunette, and Gaetano Borriello. “Video: Open Data Kit Tables”. In: *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys ’14. Bretton Woods, New Hampshire, USA: ACM, 2014, pp. 392–392. ISBN: 978-1-4503-2793-0. DOI: `10.1145/2594368.2602532`. URL: `http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2594368.2602532`.

[78] Samuel Sudar et al. “ODK Tables: Case Studies in Deployment”. In: *Proceedings of the 4th Annual Symposium on Computing for Development*. ACM DEV-4 ’13. Cape Town, South Africa: ACM, 2013, 25:1–25:2. ISBN: 978-1-4503-2558-5. DOI: `10.1145/2537052.2537077`. URL: `http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2537052.2537077`.

[79] Samuel Sudar et al. “ODK Tables : Case Studies in Deployment”. In: (2013), pp. 12–14. DOI: `10.1145/2537052.2537077`.

[80] O D K Team. Private Communication. 2015.

[81] Kentaro Toyama. *Geek heresy: Rescuing social change from the cult of technology.* 2015.

[82] *Twitter Fabric, https://dev.twitter.com/products/fabric.* Dec. 2014. URL: `https://dev.twitter.com/products/fabric`.

[83] Aditya Vashistha et al. "Sangeet Swara: A Community-Moderated Voice Forum in Rural India". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.* CHI '15. Seoul, Republic of Korea: ACM, 2015, pp. 417–426. ISBN: 978-1-4503-3145-6. DOI: `10.1145/2702123.2702191`. URL: `http://doi.acm.org/10.1145/2702123.2702191`.

[84] *Vine Trust.* `http://www.vinetrust.org/`. 2017.

[85] *Web-Server-Chrome.* 2017.

[86] Alec Wolman et al. "On the Scale and Performance of Cooperative Web Proxy Caching". In: *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles.* SOSP '99. Charleston, South Carolina, USA: ACM, 1999, pp. 16–31. ISBN: 1-58113-140-2. DOI: `10.1145/319151.319153`. URL: `http://doi.acm.org/10.1145/319151.319153`.

[87] Ka-Ping Yee. "Aligning Security and Usability". In: *IEEE Security and Privacy* 2(5) (Sept. 2004), pp. 48–55. ISSN: 1540-7993.

[88] Mariya Zheleva et al. "The Increased Bandwidth Fallacy: Performance and Usage in Rural Zambia". In: *Proceedings of the 4th Annual Symposium on Computing for Development.* ACM DEV-4 '13. Cape Town, South Africa: ACM, 2013, 2:1–2:10. ISBN: 978-1-4503-2558-5. DOI: `10.1145/2537052.2537060`. URL: `http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2537052.2537060`.