THESIS

INFORMATION COLLECTION USING HANDHELD DEVICES IN UNRELIABLE NETWORKING ENVIRONMENTS

by

Marisol M. Torres

June 2014

Thesis Advisor:	Gurminder Singh
Second Reader:	Arijit Das

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2014	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE INFORMATION COLLECTION USING HANDHELD DEVICES IN UNRELIABLE NETWORKING ENVIRONMENTS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Marisol M. Torres	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Information collection is a critical task in a wide range of tactical and strategic military and civilian operations. The ability to gather accurate information in a timely manner is a difficult task, but sharing that information and making it accessible in near real-time compounds the challenges. The ability to do all three in an environment where network connectivity is intermittent or non-existent seemed virtually impossible until recent years.

Network connectivity plays a significant role in how we conduct business. Our requirements to share information do not change simply because there is little or no existing infrastructure in our area of operations. In fact, information sharing can prove to be more important in areas where infrastructure is simply nonexistent. It is for this reason that the Lighthouse suite of applications was conceived.

This research takes a deeper look at specific goals and requirements for information collection using handheld devices, and identifies a data synchronization algorithm and existing network connectivity technologies that can be used to implement sharing between handheld devices. The recommendations can be used to enhance information sharing, promote accuracy of data, and improve the efficacy of information-gathering techniques when implemented in austere networking environments.

14. SUBJECT TERMS Information Collection, Challenged Networks, Immature Networks, Unreliable Network Connectivity, Disconnected Networks, Intermittent Network Connectivity, Delay Tolerance, Wireless Networks, Wi-Fi Direct, Near Field Communication, Direct Bluetooth Connection, Handheld Devices, Data Sharing, Peer-to-peer Network Database Synchronization	**15. NUMBER OF PAGES** 89
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

INFORMATION COLLECTION USING HANDHELD DEVICES IN UNRELIABLE NETWORKING ENVIRONMENTS

Marisol M. Torres
Captain, United States Army
B.S., Mount Saint Mary College, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2014

Author: Marisol M. Torres

Approved by: Gurminder Singh
Thesis Advisor

Arijit Das
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Information collection is a critical task in a wide range of tactical and strategic military and civilian operations. The ability to gather accurate information in a timely manner is a difficult task, but sharing that information and making it accessible in near real-time compounds the challenges. The ability to do all three in an environment where network connectivity is intermittent or non-existent seemed virtually impossible until recent years.

Network connectivity plays a significant role in how we conduct business. Our requirements to share information do not change simply because there is little or no existing infrastructure in our area of operations. In fact, information sharing can prove to be more important in areas where infrastructure is simply nonexistent. It is for this reason that the Lighthouse suite of applications was conceived.

This research takes a deeper look at specific goals and requirements for information collection using handheld devices, and identifies a data synchronization algorithm and existing network connectivity technologies that can be used to implement sharing between handheld devices. The recommendations can be used to enhance information sharing, promote accuracy of data, and improve the efficacy of information-gathering techniques when implemented in austere networking environments.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
 A. MOTIVATION 2
 B. OBJECTIVES 3
 C. RELEVANCE TO THE DEPARTMENT OF DEFENSE 3
 D. THESIS OUTLINE 3

II. BACKGROUND 5
 A. NORMAL DATABASE OPERATIONS 5
 B. DEGREES OF CONNECTIVITY 5
  1. Constant Connection 6
  2. Weak Connection 7
  3. Intermittent Connection 7
 C. DIFFERENT MODELS 7
  1. Thin-Client with Remote Access to Data 7
  2. Device-to-Master 9
  3. Publisher-Subscriber 11
  4. Peer-to-Peer 12
  5. Ad Hoc Sensors 12
 D. PREVIOUS SOLUTION 13
  1. The Field Information Support Tool 13
   a. System Components 15
   b. FIST Goals 15
  2. Lighthouse, Version 1.0 16
   a. System Components 17
   b. Use Cases 17

III. LIGHTHOUSE: THE NEXT GENERATION 19
 A. GOVERNMENT-OFF-THE-SHELF SOLUTION 19
 B. DISCONNECTED TOOLS 19
  1. Interface for Form Creation and Integration 20
   a. Separate / Disparate Data 20
   b. Stand-Alone Tools 20
  2. Data Sharing without Network Connectivity 21
   a. Near Field Communication (NFC) 21
   b. Bluetooth 23
   c. Wi-Fi Direct 24
 C. DEVICE INDEPENDENCE 24
 D. PROPOSED CAPABILITIES 25
  1. Form 26
  2. Map 26
  3. Visual 27
  4. Photo Gallery 28
  5. Help / Codebook 29
  6. Other Capabilities 29
   a. Baseball Card 29

b. New Links ... *30*
c. Export Capability ... *31*
d. Database Integration ... *31*

IV. METHODOLOGY ... **33**
A. UPDATE USING FILE EXPORTS ... **33**
1. Use Case ... **34**
B. DATABASE SYNCHRONIZATION ... **38**
1. Data Sharing Algorithm ... **38**
2. Receiving Data Algorithm ... **40**
3. Algorithm Walk-Thru ... **41**
4. Implementation ... **43**
C. DATABASE STRUCTURE ... **58**
1. Sample Local SQLite Database Tables ... **59**
2. Sample Database Creation Script ... **60**
3. Chapter Summary ... **61**

V. CONCLUSIONS ... **63**
A. MAJOR ACHIEVEMENTS ... **63**
B. FUTURE WORK ... **63**
1. Seamlessness ... **63**
2. Connectivity ... **64**
3. Off-line Maps ... **64**
4. Data Visualization ... **65**
5. Device Independence ... **66**

LIST OF REFERENCES ... **69**

INITIAL DISTRIBUTION LIST ... **71**

LIST OF FIGURES

Figure 1. Sample with Central Database Server (after MobiForms 2014) 8
Figure 2. iTunes UML Deployment Diagram (from iTunes 2014a) 10
Figure 3. Typical Publisher / Subscriber System (from Oracle 2014) 11
Figure 4. Peer-to-peer network of mobile devices (after MobiForms 2014) 12
Figure 5. Diagram of FIST Components (from Longley 2010a, 3) 14
Figure 6. Possible Lighthouse “Home” and “Help” Screens 26
Figure 7. Possible “Form” and “Map” Screens 27
Figure 8. Possible “Visual” and “Photo Gallery” Screens 28
Figure 9. Possible “Baseball Card” View (after Vectorcharacters 2014) 30
Figure 10. Sharing Data Flowchart 42
Figure 11. Receiving Data Flowchart 43
Figure 12. Start of Program Action Sequence of NPS5 44
Figure 13. Steps 3 and 4 of NPS5 45
Figure 14. Final Step of NPS5 46
Figure 15. Start of Program Action Sequence for NPS1 47
Figure 16. Screens 3 and 4 of NPS1 48
Figure 17. NPS5 screen after receiving data from NPS1 49
Figure 18. NPS5 sharing data with Galaxy Nexus 50
Figure 19. NPS5 receiving data from Galaxy Nexus 50
Figure 20. NPS5 after inserting data from Galaxy Nexus 51
Figure 21. Galaxy Nexus sharing data with NPS5 51
Figure 22. Galaxy Nexus after sharing data with NPS5 52
Figure 23. Galaxy Nexus sharing data with NPS1 53
Figure 24. Galaxy Nexus after sending data to NPS1 53
Figure 25. Galaxy Nexus after receiving data from NPS1 54
Figure 26. NPS1 receiving data from Galaxy Nexus 54
Figure 27. NPS1 sending data to Galaxy Nexus 55
Figure 28. NPS1 after sharing data with Galaxy Nexus 55
Figure 29. NPS5 Database Records 56
Figure 30. NPS1 Database Records 57
Figure 31. Galaxy Nexus Database Records 57
Figure 32. Sample SQLite Data Structure 58
Figure 33. Sample Map View Using Leaflet (from Leaflet 2014) 65
Figure 34. Sample Link and Tree Diagram (from MindFusion 2014) 66

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. List of Devices Tested 44
Table 2. Emp Table 59
Table 3. Sample Person Table 59
Table 4. Sample Person Data Table 60
Table 5. Sample Photo Table 60

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AOR	area of operation
API	application programming interface
ATM	automated teller machine
ATMC	automated teller machine controller
BLOB	binary large object
C-IED	counter-IED operations
CIDNE	Combined Information Data Network Exchange
CN	counter-narcotic
COIN	counterinsurgency operations
CORE	Common Operational Research Environment
COTS	Commercial-off-the-shelf
DOD	Department of Defense
DSL	digital subscriber line
FIST	Field Information Support Tool
GPS	Global Positioning System
HA/DR	humanitarian assistance and disaster response
HTS	Human Terrain System
IED	improvised explosive device
ISVG	International Studies of Violent Groups
JLOC	joint logistics operations center
JMS	Java Message System
JUNG	Java Universal Network/Graph Framework
KTG	Kestral Technology Group
MILOB	military observer
NATO	North Atlantic Treaty Organization

NFC	near field communication
NPS	Naval Postgraduate School
ODK	Open Data Kit
ORA	Organizational Risk Analyzer
OS	operating system
PS	personal computer
RF	radio frequency
RFID	radio frequency identification
SDK	software development kit
SQL	structured query language
UML	unified modeling language

ACKNOWLEDGMENTS

First, I want to thank my family. Having your love and support made all the difference in the world, and I could never thank you enough. To my son Samuel who has had to endure many years of separation because of my military service, I understand that these times apart were difficult. I am proud of the young man you have become despite those difficulties. To my love, Jamar, thank you for helping me through the daily grind, countless hours of proofreading, exam preparation, as well as home cooked meals when I found myself stretched thin. Thank you to my sisters, Jake and Bren, for the late night phone calls and text messages. Those little pep talks were much needed to maintain the motivation required to complete this task in conjunction with the loss in the family. Thanks to my fellow students, specifically in Networks and Mobility, for the many ideas and approaches to problem solving. Thanks to Chuck Casey for your help in the last few weeks of prototype development. Finally, I would like to thank my thesis advisors for continuing to push me to complete tasks in support of this project, ensuring that I had access to everything I needed to make this project a success. Your guidance was critical to completion of this project.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The Common Operational Research Environment (CORE) Lab at the Naval Postgraduate School (NPS) developed the Lighthouse project that allows intelligence analysts and tactical operators to visualize the battlefield by providing a map of the human terrain (Longley 2010a, 1). The human terrain system is a military intelligence program used to better understand the local population (U.S. Army Human Terrain System 2014). Understanding the human terrain takes into account sociology, linguistics, political science, anthropology, and other social sciences (*Wikipedia* 2014a).

Lighthouse was developed at NPS by two students and has been adapted and implemented in several use cases (Honeggar 2014). The inspiration for the project came from difficulties faced in data collection and analysis problems encountered in Iraq and Afghanistan. It is typical of information collectors to gather large amounts of data during normal operations. While the services were able to capture vast amounts of data, much of the information was unstructured text or in written reports. This information overload made content organization and management difficult. Oftentimes, information was not exploited because of time constraints. Lighthouse is designed to capture structured data in the field for rapid exploitation by analysts while enabling powerful analytical methodologies to be applied against the data to reduce the uncertainty involved in decision making (Honeggar 2014). Lighthouse exploits the advanced capabilities of computer and mobile technology to improve the process and reduce the time and effort required for data collection and analysis.

Today’s smartphones and tablets afford incredible storage and processing capabilities. The various applications that make up Lighthouse have enhanced collection methods by enabling personnel to gather pertinent information using predefined forms stored on mobile phones. The data collected is stored on the device and can be sent to an email address or uploaded to a website. Other tools within the Lighthouse project are used to aggregate the data in order to provide more information to intelligence analysts.

This research analyzes the current Android-based Lighthouse suite of applications that focus on data collection tools currently employed by the system. It determines the requirements for implementing an improved mobile application for data collection. These guidelines would help in creating an improved application that can be deployed on virtually any mobile device platform, as well as most web browsers on any computer operating system. Using popular, open-source software development kits (SDK) and application programming interfaces (API), the prototype can be developed to detect the type of device on which it is running and take advantage of additional screen real estate, processing capabilities of larger devices, or computer processors.

A. MOTIVATION

Lighthouse is used by several government agencies to improve data collection and analysis processes. A current limitation is that Lighthouse currently requires a constant broadband connection to upload data. This connection is often not available due to geography and used too much battery power.

Another limitation is that special equipment is required for information to be properly shared in those austere networking environments. Wave radios are one means of gaining the required network access for collectors to upload their data. Persistent Systems[1] is one company that provides this type of equipment but each radio can cost up to $5,000 per device. Depending on how critical the information being collected is, this may be a small price to pay.

Mobile devices, however, have sensors that could possibly remove the requirement for wave radios or other equipment. This research would lead to an improvement in the functionality of the overall system as well as a cost savings by offering a solution that removes the requirement for specialized equipment.

[1] Persistent Systems offers a Wave Radio product line that has mostly been used by government entities, but has expanded to industrial companies. For more information, visit www.persistentsystems.com.

B. OBJECTIVES

One capability that will make Lighthouse more useful is to make it more reliable in various wireless networking conditions that occur in real-life, without the requirement of specialized military equipment. The ultimate goal of this thesis is to enable database synchronization between data collectors in remote environments without Internet or Broadband connectivity. Sharing data in those remote environments can enhance data collection and analysis operations

C. RELEVANCE TO THE DEPARTMENT OF DEFENSE

The original inspiration for development of Lighthouse was to streamline and improve data collection and analysis specifically in austere environments such as the areas of operation (AOR) encompassed by the Global War on Terror. Lighthouse was used in remote regions of Iraq and Afghanistan with special military equipment because of the lack of network infrastructure. This enabled sharing of data from the remote sites to the rear headquarters. The insights gained through this research are impactful in various areas throughout the DOD to include improved collection capabilities without the use of specialized equipment, which also results in a significant cost savings to American taxpayers.

D. THESIS OUTLINE

Chapter I introduced the research topic and objectives. Chapter II describes the evolution of Lighthouse from Field Information Support Tool (FIST) into its current form. It explains the motivation and goals hoped to be accomplished. It also discusses some of the shortcomings of the current implementation of Lighthouse. Chapter III describes several approaches that could be used to improve current operations. Chapter IV describes the methodology, results and the challenges faced in developing the solution and issues with database synchronization, especially when in a disconnected environment. Chapter V summarizes this research and future work to be conducted.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter discusses some background information that is helpful for readers prior to moving forward. Familiarity with database operations is not critical, as most of the difficulties involved in database sharing and synchronization are fairly simple concepts to understand. Additionally, levels of network connectivity and the types of applications recommended based on connectivity is explained within this chapter. Finally, the some research on the FIST is detailed to introduce the origins of Lighthouse to provide readers with the proper frame of reference.

A. NORMAL DATABASE OPERATIONS

The vast majority of current database systems employ client-server architectures, requiring a connection to a main database stored on a server. The abrupt spike in mobile applications and mobile devices in general created a requirement for a closer look into how database operations in the mobile realm differs from previous standards of database operations. Since mobile device capabilities have increased so dramatically, users expect the same kind of growth in their ability to access pertinent data. For example, users were once satisfied with their ability to access their bank account balances by driving to the nearest automated teller machine (ATM). However, these days, users who cannot obtain this information via the Internet on a personal computer (PC) or mobile phone would see this as a serious limitation and would likely switch to a bank that offers this convenience.

B. DEGREES OF CONNECTIVITY

At the advent of the Internet, protocols were developed to deal with many issues such as connectivity and data-replication to ensure accessibility and reliability of data, as well as synchronization to establish rules to ensure accuracy of data when seemingly simultaneous updates take effect. For example, the first ATM did not disperse cash, rather only received deposits. As their capabilities and popularity increased, ATM machines could be connected via a dial-up connection, digital subscriber line (DSL) or ATM controller (ATMC) (*Wikipedia* 2014b). A husband and wife accessing the same account via different ATMs at the same time could cause the data to become out of sync with the main account database. Banks had to implement procedures to ensure that there were no errors by replicating the account at the start of the transaction and verifying whether there was a change in that data prior to making an actual change to the database.

A slow ATM connection could mean several changes to the account prior to completion of a single transaction. This is a well-known race-condition problem solved many years ago in the days of mainframe servers. It goes without saying that network connectivity is a critical factor for access to data, as well as reliability of that data. Yet, we can also see that certain types of operations can tolerate different degrees of connectivity. The biggest challenge to connectivity, especially in mobile settings, is that one cannot guarantee connectivity at all times, and a drop in connection can cause data synchronization issues that must be handled.

The degree to which a device must remain connected has a significant impact on the flexibility and usability of the device. Maintaining "constant" connections to the main database can become extremely expensive and cumbersome, and given the current state-of -the-art of deployment of mobile networks, it cannot be taken for granted.

1. Constant Connection

Mobile applications that require constant connectivity to a database or other server can be rendered useless if a connection is unavailable. As a result, that application becomes useless to the device user since the application will simply not function properly without connectivity. For this reason, developers must be careful to prevent making connectivity a constant requirement. It may be appropriate in some cases, but for many applications, constant connectivity is not truly required.

Another challenge with the requirement for a constant connection is the fact that service interruptions cannot always be avoided or predicted in advance. So, developers must ensure they account for disruption at every stage of any transaction. They must deal with actions that must occur after connectivity is re-established in order to correct any potential discrepancies caused by the interruption (Terry 2008).

2. Weak Connection

Weakly connected systems can communicate through low-bandwidth or high latency connections. However, there can be a severe impact if connection drops during transaction. The ideal algorithms for this environment must be conservative. The algorithm introduced in this thesis can be used in this environment because it avoids having unnecessary re-transmissions.

3. Intermittent Connection

An intermittently connected system takes into account or expects periods during which it cannot communicate. The major difference between this and weakly connected systems is that applications designed for intermittently connected systems take into account the fact that connectivity is not guaranteed (Terry 2008). This means that the applications attempt to gather and hold transactions until the next connection is made. In order to ensure that transactions are not lost in the event of a connection dying in the middle of a transaction or prior to completion of waiting transactions, the application do not totally remove transactions that were recently completed. Rather, upon re-connection, the application verifies that the previous transactions were successful prior to considering the transaction complete, and removing them from the pending transactions queue.

C. DIFFERENT MODELS

Various models have been developed which take into account the degree of connectivity and other requirements of the application involved. As mentioned previously, a disruption in connectivity can have significant impact on the consistency of data. This section will detail the requirements for replication of data in some common models in use today.

1. Thin-Client with Remote Access to Data

One of the predominant models used is storing data on a server that will be accessed by mobile devices. Figure 1 is the common architecture employed by developers and providers to allow their mobile clients access to centralized data. It depicts three different types of mobile devices that connect wirelessly to a database server. The actual backend database is not important to the mobile clients. They simply want to be able to access their data. The mobile devices do not handle sharing content either. In fact, neither is concerned with any connection other than its own (Long 2011).

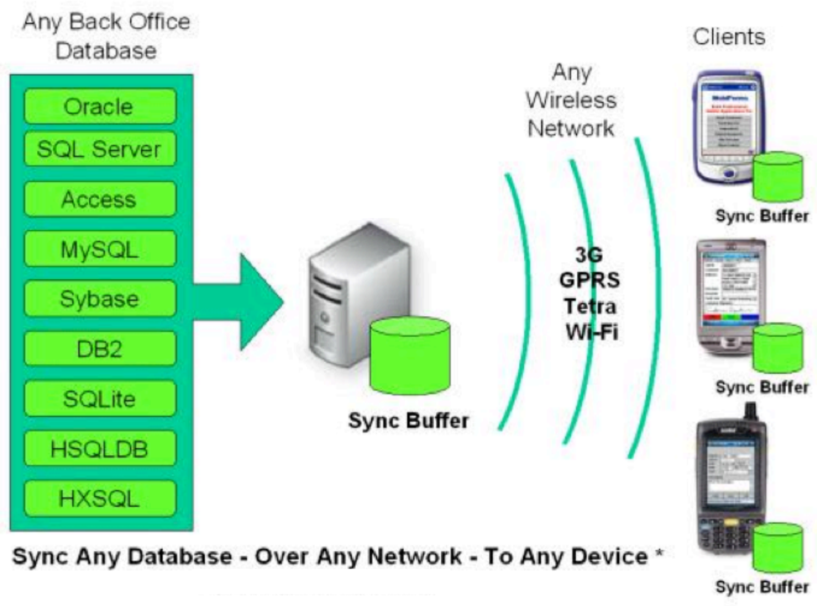


Figure 1. Sample with Central Database Server (after MobiForms 2014)

In implementations where there is no sync buffer (or replica) residing on the local device, the application will not be able to run because it cannot locate the data required. Other implementations employ a buffer that can cache some data locally. After an initial connection between the mobile device and the remote sync buffer, the device creates a local replica. This prevents the application from failing to function due to lack of wireless connectivity (Terry 2008). As long as a connection exists, the local information can remain in sync with the remote data. In case of disconnection, requests for data will access the local data. If these devices require the ability to modify the remote database, the devices make modifications to the local buffer and post updates to the remote server when a connection to the remote data server becomes available. Mobile devices have limited battery life, so a constant connection can consume too much power, rendering the device useless in environments where recharging a battery can be difficult. The local sync buffer enhances power saving by only requiring a connection when an update must be posted, or the mobile device user requests updates from the database (Nori 2007).

2. Device-to-Master

In many cases, mobile devices are used for functions that require data from a master database or server to be replicated on the local device itself. One common example is an iPod or iPad, but the same applies to other brands of mp3 players and tablets. These mobile devices range in size or capacity. Users have the ability to modify the contents of their iPod. They can do so without having to re-purchase content previously purchased. Today, they might load a movie on their device and decide next week to put more music on the device instead. This flexibility is one of the features that make these devices attractive to users. If they did not have the ability to modify the content, users would not feel they are worth the cost.

Figure 2 provides a visual depiction of this architecture used by iTunes. This is likely most familiar example of the device-to-master model, yet media services are not the only service requiring replication of data. Other uses for this type of system are management of contacts and electronic mail. While the architecture may appear to be the same as the previous description, their behavior is very different. The major difference is that thin-client systems do not necessarily require mobile devices to store or replicate any data (Terry 2008). Those devices simply access remote sites and display information the user wants to see. They do not require the device to actually store any of the information accessed. This architecture, however, requires that some amount of data is stored on the mobile device from the master.

In this model, one device is the “master” device. This means that one device acts as the authority or manager. Oftentimes, when synchronization occurs, it must occur between one device and its master. It is not often that updates are done between more than two devices at a time. If there are multiple devices that must synchronize, they do so by taking turns, synchronizing with the master in a pairwise fashion (Nori, 2007).

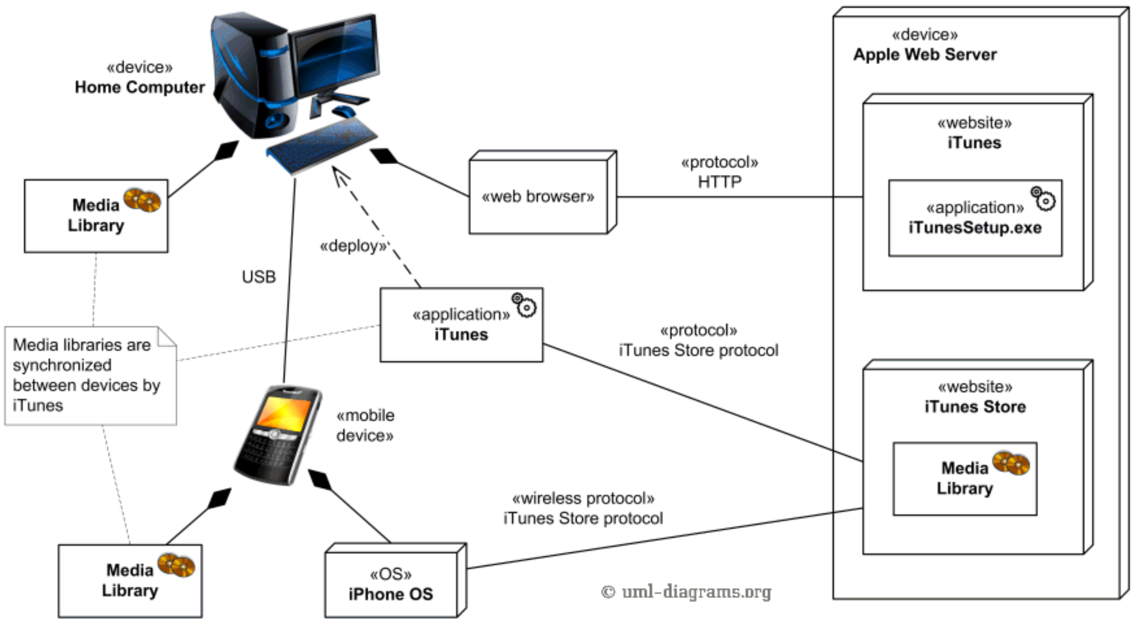


Figure 2. iTunes UML Deployment Diagram (from iTunes 2014a)

Sticking with the music example, a user would connect their iPod to their computer. The computer may or may not have the entire user's content stored locally. So, that computer would connect to the Internet to access their service provider. In the case of an iPod, that provider would be Apple's iTunes Store. The iTunes Store maintains the user's account, tracking any purchases and content owned or authorized to be loaded onto the user's device (iTunes 2014b). Users are given the opportunity to reload their devices at will. Also, they can synchronize purchases made by the same user from different devices.

Proper management of user content is critical for providers to maintain loyal customers. If users start to experience diminished service because of the providers' practices, the likelihood that those users will seek more reliable service increases. No user wants to log in one day and see their entire music library has disappeared. Additionally, users want their experience to be enhanced as much as possible. Therefore, the simpler it is for them to perform common tasks without experiencing errors, the better. For this reason, providers attempt to make common functions simple, or automatic. They allow users to perform many tasks in background processes so that users need not be inconvenienced. These principles apply to virtually all user content, not just music.

3. Publisher-Subscriber

Publisher-subscriber systems are defined by their behavior. Publishers publish bits of information that are broadcasted to their subscribers. Therefore, subscribers subscribe to receive specific types of information from specific publishers. Either party can be mobile or fixed, wireless or wired devices (Terry 2008).

Two simple examples of this type of system are sports or financial updates. Sports fans might subscribe to receive scores or other information about their favorite team from a specific team or sports channel. Stock market investors may wish to receive periodic updates from their investment firms, financial institution or other provider. Weather, local news and traffic alerts are other forms of publisher-subscriber systems that are extremely popular for mobile subscribers using phones, smart watches, GPS, or satellite radios.

Figure 3 shows a typical publisher/subscriber system. This case specifically depicts the Java Message System, a Publisher/Subscriber system that is explained in greater detail at either Oracle's JMS site (Oracle 2014) or the JMS Wikipedia page (*Wikipedia* 2014c). Client 1 is the publisher of a topic that Clients 2 and 3 have both subscribed. Upon publishing of the content, the topic is delivered to both clients.

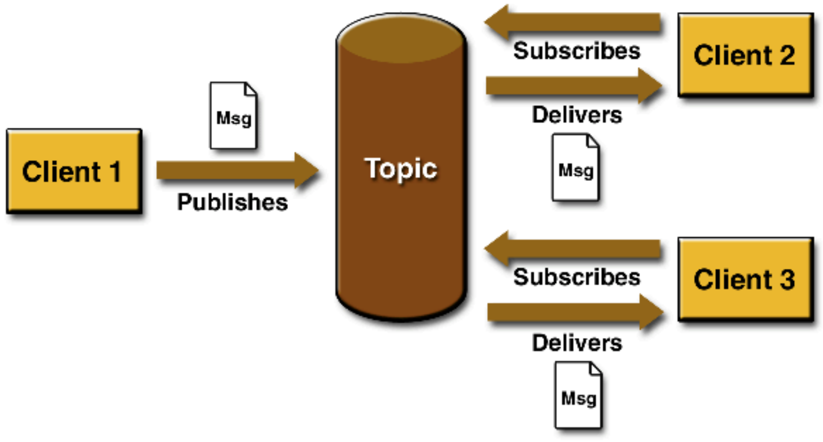


Figure 3. Typical Publisher / Subscriber System (from Oracle 2014)

4. Peer-to-Peer

In true peer-to-peer systems, no device acts as the master. Peers can connect to and update one-another locally. They need not be connected through a central server or even have Internet access. This type of system is ideal in a weak, intermittently connected or disconnected environment. This is the major advantage of this type of system; however, synchronization of data could be an issue if developers are not careful about decentralized conflict resolution (Terry 2008).

Figure 4 depicts a peer-to-peer local network where mobile devices share information in their databases with one another. The database can either be a distributed database where each device holds different portions of the entire database. It can also represent a system that starts with one database that is replicated on each mobile device and updates are replicated to the other devices as needed.

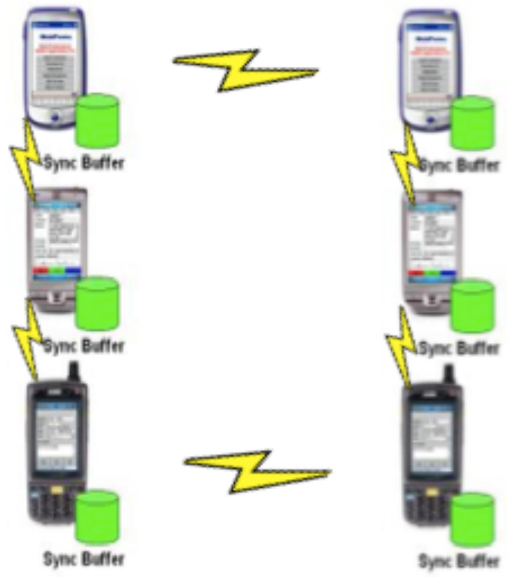


Figure 4. Peer-to-peer network of mobile devices (after MobiForms 2014)

5. Ad Hoc Sensors

The final model we discuss is Ad Hoc Sensors. Sensor networks are mesh networks comprised of nodes that gather certain information or data. For example, one might have a network of cameras with motion sensors that start recording when the device senses motion. Each node has wireless connections between them and may share data between each device on the mesh network. Another example could be temperature sensors connected to one another wirelessly, ultimately, providing the information to a PC. These devices could be used as an advanced warning system for a severe weather or volcano eruptions. Motion sensors with cameras attached could be integrated into a mesh network as a wireless fence and detect a perimeter breach, activating the camera to take images. The captured images can be sent to a PC on which a person would verify whether the breach was made by a person, animal or strong gust of wind.

D. PREVIOUS SOLUTION

Commercial-off-the-shelf (COTS) solutions to solve technical problems within the military are commonplace. The military procurement system can take several years and billions of dollars to develop and implement specific military equipment needed for a specific mission. Making slight modifications to commercial solutions is one method of significantly reducing taxpayer costs.

FIST is a COTS system comprised of commercially available smartphones, customized software, and a commercial backend or portal for information management. (Longley 2010a) The system was used to enable information collection in remote environments using commercial smartphones, right out of the box. This enhanced collection efforts by simply making it easier to gather pertinent information, thereby increasing the number of reports collected. This increase required more robust means of intelligence analysis, since the speed of collection was so significantly improved.

1. The Field Information Support Tool

The Field Information Support Tool (FIST) started out as a project to integrate cellular technology into the Army's tactical radio frequency (RF) networks. Eventually, it grew into a means of gathering information using a smartphone application and sharing that information more readily, in any environment (Longley 2010a). It was an entire suite of commercial capabilities that was used to improve intelligence operations. It enabled intelligence operators to collect, save and forward reports more easily, and allowed analysts to view and conduct their analysis in a more timely manner.

Some areas in which FIST has proven its usefulness include military counterinsurgency and counter-IED operations, humanitarian assistance and disaster recovery operations, anti-gang as well as anti-drug operations (Longley 2010a). Figure 5 shows a diagram of FIST Components. FIST incorporates a method for collection of data, transferring that data to a central server where intelligence analysts could conduct geospatial, link, temporal, and social network analysis, and information exchange with external data to include the Combined Information Data Network Exchange (CIDNE) and the International Studies of Violent Groups (ISVG).

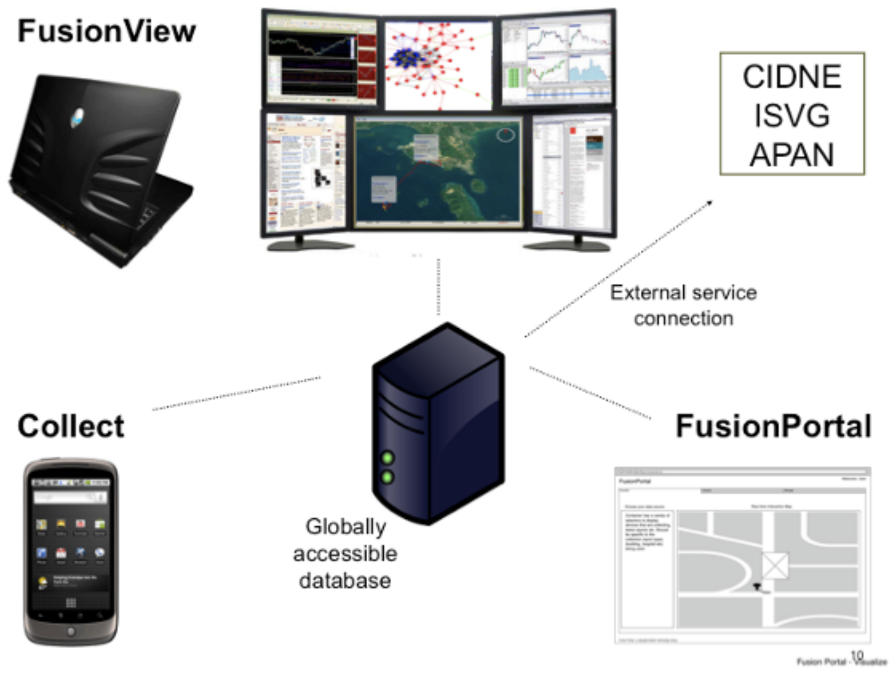

FusionView
CIDNE
ISVG
APAN
External service connection
Collect
Globally accessible database
FusionPortal
Fusion Portal - Visualize
10

Figure 5. Diagram of FIST Components (from Longley 2010a, 3)

a. System Components

FIST consisted of several components known as *Collect*, *FusionPortal* and *FusionView*.

(1) *Collect* was an Android-based application used to collect data in the field and transfer data the central database. It used forms and a "store and forward" method of information delivery. Information was collected in very remote, poorly connected environments using smartphones. In the event that network connectivity was dropped, reports were stored locally and sent upon re-establishment of connectivity.

(2) *FusionView* was a system designed by Kestral Technology Group (KTG) in conjunction with the Naval Postgraduate School to aggregate data collected. It provided a means of data visualization, asset tracking, dynamic report updating, alerts and alarms that could be viewed on laptops, handhelds or network clusters (Longley 2010a, 8).

(3) The *FusionPortal*, also designed by KTG, provided a means for integration with commercial software for further analysis. It was a web-based management system that worked with: UCINET, a social network and cultural domain analysis tool developed by Analytic Technologies; Pajek, another social network analysis program that provides visualization and analysis that compliments UCINET; the Organizational Risk Analyzer (ORA), is a network assessment tool developed at Carnegie Mellon that examines changes in networks over time, and identifies key players and vulnerabilities; ArcInfo, a geospatial analysis tool widely used throughout the Department of Defense (DOD) (Longley 2010a, 13). The system also allowed for simple integration with the Worldwide Civil Information Database, the International Studies of Violent Groups and the Combined Information Data Network Exchange (CIDNE) (Longley 2010b).

b. FIST Goals

The main reason for development of a "new" system or adopting a new way of doing business is generally for a perceived improvement or savings in resources and/or time. FIST was able to accomplish various improvements.

(1) Rapid prototyping was one of the reasons for choosing to go with COTS equipment rather than purpose-built equipment catered to the army for a specific need. The COTS approach has obvious advantages. The main savings was in the cost and time it normally takes to develop a military or government hardware solution. By customizing some of the existing smartphone applications instead of developing a military smartphone, development costs were significantly reduced. This allows for rapid adaptation and testing of commercial smartphone applications (Longley 2010a).

(2) Ease of training was another critical reason for adapting FIST. Many military solutions involve time and cost intensive training and maintenance modules. Since smartphones have become ubiquitous and users are already familiar with smartphone usage in their personal lives, training operators to use a smartphone form is a fairly trivial task. Virtually no training has to happen for an operator or information gatherer to be able to use a form based application. The most difficult part of FIST seemed to be training a "form manager" to create the different forms to be used for gathering the critical information.

(3) Improved data collection and analysis was also accomplished by enabling the collected information to be shared easily. That rapid sharing cut down the overall analysis time, simply by allowing the analysts to obtain the information collected more quickly than usual. The application also enabled the data to be stored or exported in several different formats. This automated conversion made it easier for intelligence analysts to use other commercial analysis tools in their analysis (Longley 2010a).

2. Lighthouse, Version 1.0

As previously stated, the original concept resulted in the creation of FIST. As the tool became more popular, capabilities expanded and other tools were integrated into the system, it became more commonly known as the Lighthouse Project. It can be thought of as a living and breathing system, as other tools are constantly being researched and integrated into the system as needed.

a. System Components

(1) Open Data Kit (ODK) is a free, open-source data collection tool-kit that was developed by the University of Washington's Department of Computer Science and Engineering (ODK 2014a). It enables organizations to create, implement and manage their very own mobile data collection systems. Using the tools within ODK, an organization can build their own forms that are loaded onto smartphones or other mobile devices. Once loaded, that device can be used to collect virtually any type of data in the field, including location information, audio-visual, photos, barcodes, or just old-fashioned text data. Of course, none of this is extremely useful without the aggregation tool that is also included. According to their website, it uses Google's infrastructure and local servers with MySQL and PostgreSQL on the backend (ODK 2014b).

(2) Google Fusion Tables are used to do basic link analysis and create line, bar and pie graphs (Longley 2012a). The data can be exported into comma-separated values file format. These visual representations are sent to remote data collectors and decision makers.

(3) Organizational Risk Analyzer (ORA) remains a large part of Lighthouse. It has powerful, dynamic network analysis capabilities based on the specific data that is uploaded to the server. According to the Lighthouse training website, "

> ORA's approach is the idea of a meta-matrix of networks that not only includes social networks but knowledge networks (who knows what), information networks (what ideas are related to what), assignment networks (who is doing what), need networks (what knowledge is needed to do the task) and so on. (Longley 2012b)

The ability to export the information into various file formats makes it easier to move information into the different tools and obtain the required data analysis.

b. Use Cases

(1) Law Enforcement Agencies have been using Lighthouse tools for counter-gang and counter-drug operations. Specific information about known and suspected drug dealers or gang members enables officers to perform social network analysis that has proven invaluable to combat these well organized criminal organizations. Not only is the

information viewable on a map in order to overlay different events and people, but it is also able to look deeper and create possible links for officers to investigate further.

(2) Humanitarian Assistance and Disaster Recovery operations can be significantly enhanced using Lighthouse. In fact, it was used to collect survey information following the earthquake in Haiti, by a team from the University of Central Florida Institute of Simulation and Technology.

(3) Counterinsurgency and counter-IED operations are two areas where the urgency of information collection cannot be overstated. The most recent information tends to be the most critical, and sharing this information among collectors is also critical. Using social network analysis tools and modifying the type of information gathered, counter-IED operations are tremendously enhanced. Intelligence personnel can now analyze component similarities, attributes and other characteristics that can shed some light on the various bomb-making networks in various regions of the world.

III. LIGHTHOUSE: THE NEXT GENERATION

A. GOVERNMENT-OFF-THE-SHELF SOLUTION

Government-off-the-shelf (GOTS) systems are generally COTS systems that are created, modified, developed and owned by the government. Usually these tools are developed by the technical staff of government agencies with a requirement to fill. These systems can often be shared between different government agencies without additional costs involved. This is usually a more inexpensive approach to fulfilling a requirement than contracting out a job. Also, using open-sourced programming tools can help facilitate programming and development needs. Catering open source programs to suit the needs of Lighthouse users could streamline the tool rather effectively. The CORE Lab created a working group of Lighthouse users to determine what improvements should be made to the current system.[2] A GOTS solution might be something to consider. This chapter describes many of the ideas discussed during the working group discussions to enhance Lighthouse.

B. DISCONNECTED TOOLS

In its current form, Lighthouse has several disjointed tools that are used for one main purpose. That is, to enhance data collection and intelligence analysis efforts, especially in austere environments (Longley 2010a, 1).

When using the current system, users cannot help but notice that the environments, or graphical user interfaces used, vary from one function to another. In fact, users cannot easily switch between the different missions or tools within Lighthouse. Each tool behaves differently from the other since they were all created by different companies and with different goals in mind. With the working group's goals in mind, and recognizing that many open-source tools already exist that can enhance operations, there is no need to totally re-invent the wheel. Many of these tools can be combined into an

[2] Members of the Lighthouse working group: MAJ Sam Kemokai (USA, SF), CW4 Chad Machiela (USA, SF), Ms. Linel McCray (Monterey County Health Department), and Massachusetts State Trooper Stephen Gregorcyk

updated version of Lighthouse that would give the system the look and feel of an integrated system, rather than the disjointed system that is currently in use.

1. Interface for Form Creation and Integration

One of the major functions of Lighthouse is the ability to create forms based on an agency's needs. Each mission would typically require a different form to collect the desired information pertinent to the mission. This is an extremely powerful function. The original system used Zerion's iFormbuilder to view, sort filter and export data for analysis (Longley 2011). By 2012, it transitioned to ODK. However, there are two fundamental flaws inherent in the way this tool is currently used.

a. Separate / Disparate Data

One challenge is that each mission is separate. This means that the database created by a specific form is for a specific purpose. This data is held totally separate from other databases on the same device. Since these forms create different databases, comparing the information gathered on different forms is not a trivial task, although not impossible. This also makes it difficult to make even modest adjustments such as adding a variable. Comparing data from the two databases simply cannot be done easily on the mobile device unless the data is combined into the same, shared database. Every interface is designed for a specific mission and integrating data between the different forms is not part of the current system design. Comparisons can be done, but it requires a third interface. This type of disjointedness is not considered a positive feature.

b. Stand-Alone Tools

Another challenge is that because each tool within the Lighthouse Project is an actual stand-alone tool, each has a different look and feel. Normally, systems are designed with consistency in mind. The look and feel of a system could make or break the system. Users prefer when different screens of a specific system look and feel the same and natural to the task being performed. That is, buttons and background colors, layout and other characteristics should look and behave the same. This would make it easier for users of one tool within Lighthouse to feel comfortable with other tools in

Lighthouse. Unfortunately, since it is a set of tools from different manufacturers and developers, each tool behaves differently and interoperability can be challenging. Additionally, some tools perform similar functions but provide results in different file formats. One tool that can convert into the desired format would be the ideal solution and prevent overlapping capabilities.

2. Data Sharing without Network Connectivity

Under normal or ideal conditions, users can access nearly anything they desire as long as they have access to the Internet. Unfortunately, connectivity varies greatly from one location to the next, especially in environments in which Lighthouse is likely to be used. What happens when network connectivity cannot be taken for granted?

Lighthouse currently functions relatively simply with devices purchased through normal commercial mobile telephone retail providers. Non-sensitive data can pass through indigenous 2-4G networks or Wi-Fi hotspots if available. However, remote locations with limited infrastructure may require special equipment obtained through military contracts or other special processes. Without special equipment or additional infrastructure providing the much-needed connectivity, although collectors can continue to gather information, there is no means of sharing that data. Adding any one of the following communications capabilities would allow for collectors to share their information and render special equipment unnecessary, saving taxpayers the cost of that special equipment. The requirement for commercial or military grade data transfer methods depend on user requirements and sensitivity of data being collected.

a. Near Field Communication (NFC)

Near field communications is a method of communications that allows transfer of information between mobile devices without the use of wires (NFC Forum 2014). This is done by placing the devices in very close proximity, such as touching or within three inches. Generally, companies implement this through proprietary protocols, suited their own purposes, making it difficult for transfer between devices made by different manufacturers. There are several modes in which NFC is implemented in order to support various functions. The NFC Forum was established to standardize protocols to make

> "life easier and more convenient for consumers around the world by making it simpler to make transactions, exchange digital content, and connect electronic devices with a touch. NFC is compatible with hundreds of millions of contactless cards and readers already deployed worldwide." (NFC Forum 2014)

There are several operating modes which NFC-enabled devices are able to support.

(1) Card Emulation Mode allows NFC enabled devices to interact with external readers by simulating smart cards. This enables users to make purchases and other transactions with a simple touch.

Lighthouse collectors could use this technology to provision mission specific information in their devices with little to no effort, since this capability is already integrated into Android devices. To do so, they would simply modify the settings on their devices at the time they wish to share information and enable NFC. Then they would select the file they wish to share. Finally, they would touch each device that needs the information.

So, a team leader receives word of a short notice mission and downloads the mission pack prior to the start of the mission. The mission pack contains all of the relevant information / database for the area of operation. The team of collectors would not have to wait for every collector to download the relevant data. Rather, that time could be used to perform other essential pre-combat checks. Since using NFC and Beam is so simple, the team leader can now touch each team member's device, providing them with the relevant database information. Additionally, in other operations that are less dangerous, collectors can share their updates with one another when they meet for lunch, for example, ensuring that every team member has the most updated information.

To use this capability with Lighthouse could be beneficial, but does not really do enough to enhance Lighthouse capabilities. While this capability may be nice to have, having to be within such close proximity is extremely limiting.

(2) Peer-to-Peer Mode is used for two-way sharing of information. It uses the ISO/IEC 18092 standard and is based on the forum’s Logical Link Control Protocol

Specification (NFC Forum 2014). It can be used for devices to exchange photos, business cards, and even Bluetooth or Wi-Fi parameters.

This could benefit Lighthouse by making it easier to share connection information prior to the start of a mission. Collectors would touch devices in order to establish a list of devices to which connections are allowed, essentially creating an access list. By doing this, devices could automatically connect to "allowed" devices and automatically decline connections to devices not on the access list. These functions could happen in the background, enabling the data collector to continue gathering information rather than stopping collection efforts to verify a device or establish connectivity in the field.

(3) Reader/Writer Mode allows enabled devices to read information stored in NFC tags that are embedded on posters or other marketing displays. This mode is often used for updating frequent flyer miles or tapping to obtain special offers. It is compliant with NFC-A, NFC-B and NFC-F schemes (NFC Forum 2014).

The use of this mode in Lighthouse would be extremely limited. It could prove beneficial for inventory tracking and surveys, but is not a capability normally associated with information collection with respect to social networking.

b. Bluetooth

Bluetooth technology is another wireless standard for connecting to external devices. The major difference between Bluetooth and NFC is the range for Bluetooth is much greater. Rather than being only inches apart, Bluetooth devices can connect at distances up to 100 meters (Bluetooth 2014). Having the ability to share information between data collectors would prove to be a significant improvement to Lighthouse. This would mean that the information collected could be automatically distributed to nearby devices rather simply. These local updates would occur much more quickly than having each collector send information to the central server and downloading any changes. Data synchronization would be improved significantly using the algorithm described in the next chapter. Additionally, this local sharing would mean collectors have nearly immediate access to information updates than could be acted upon more readily.

While having the ability to share data collected by sending files between devices, it could prove more powerful to share data in the form of objects holding specific records instead. For example, the application could be programmed to automatically connect to and send data to external devices based on a specific event, as in a user has modifications or new records to send. This "automatic" attempt to connect and share data would happen without other actions taken by the data collector. It could be based on a specific amount of time, for example every fifteen minutes; or it could be based on a user having made a certain number of updates.

c. Wi-Fi Direct

A third wireless technology in growing in popularity is Wi-Fi Direct. This technology would be the most ideal when operating in remote or unreliable networking environments because it allows for the most distance between mobile devices that wish to share data. This would enable a team of collectors in remote locations to share data with one another at faster speeds, up to 250 mbps, and distances up to 200 yards (Malone 2010).

C. DEVICE INDEPENDENCE

In most cases, when applications must be changed or upgraded, software must be re-written for each type of device on which that code is supposed to run. For example, iOS devices cannot run code written for an Android device. Rather, the code must be re-written and compiled specifically for iOS devices. Several software development kits allow developers to create applications that can be run on virtually any mobile device or laptop browser, rather than writing different software for each device. Appcelerator and PhoneGap are two such software development kits that could be used to create the new and improved Lighthouse Application.

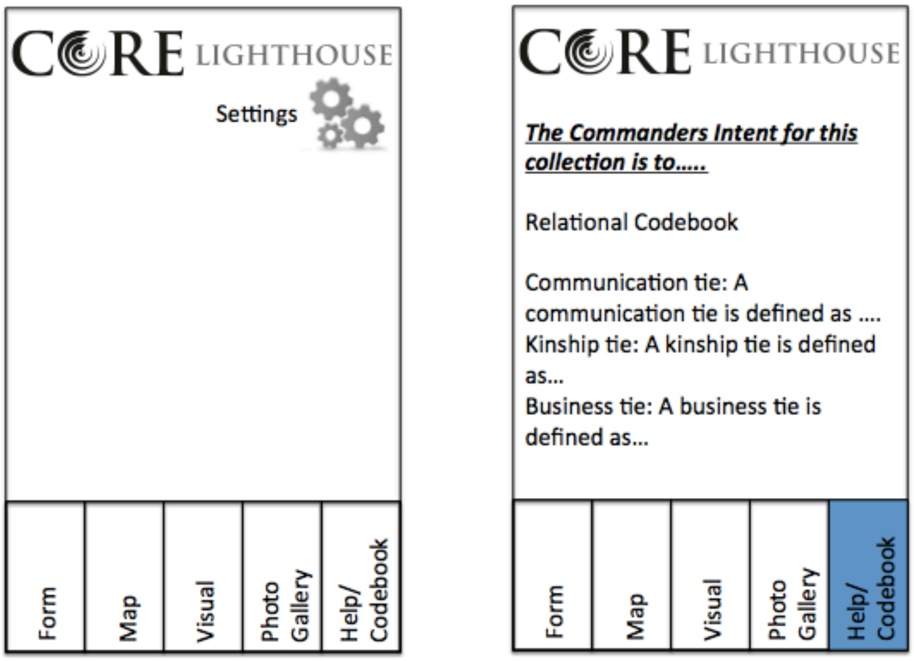
As previously mentioned, the major benefit of using one of these SDKs is that the program need only be written once. Theoretically, the code is compiled to run on any device. In fact, the user may not be able to notice a difference in how the application runs on an iPhone, iPad, Android tablet, or laptop web browser for that matter. The developer could choose to make the application look and perform in the exact same manner regardless of the device, or take advantage of additional screen real estate and processing power differences between smart phones, tablets, laptops and/or desktops.

D. PROPOSED CAPABILITIES

The next generation of Lighthouse proposes many enhancements to the current process. One of the ideas from those meetings was to develop an application that would access various features of the overall system with the click of a button. Each button or tab would serve a specific function and are discussed below. Considering that most of the software comprising the Lighthouse Project is either open-source or GOTS software, they can be consolidated into one application. The user interface can be designed to integrate various existing functions and improved upon.

Android developers use "Fragments" and "Activities" to create different views and move seamlessly between different functions of an application. Each fragment can behave as an interface for a different set of activities and serve distinct purposes. The second generation of Lighthouse could consolidate at least eight different operations for its users. These functions could be broken up into several tabs providing better flow and function to Lighthouse users. Mike Stevens, manager of the CORE Lab, created some possible screens to demonstrate the functions to be incorporated in the next Lighthouse version, after meetings with the Lighthouse working group.

The first activity would be considered the "Home" screen. There would be several tabs or buttons that would take the user to another activity of the user's choosing. This screen could look something like the screen depicted in Figure 6. This figure also depicts the "Help" screen, which displays helpful information about the application to the user. It could also display information about the mission requiring the information being collected. The highlighted or bolded tab is the active screen or activity. On the home screen, a user could select the mission by changing the settings, ultimately loading the mission database on to the remote device. From the "Home" screen, the user has the option to hit any of the tabs or activities that we call: "Form", "Map", "Visual", "Photo Gallery", and "Help".



The Commanders Intent for this collection is to.....

Relational Codebook

Communication tie: A communication tie is defined as ....
Kinship tie: A kinship tie is defined as...
Business tie: A business tie is defined as...

Figure 6. Possible Lighthouse "Home" and "Help" Screens [3]

1. Form

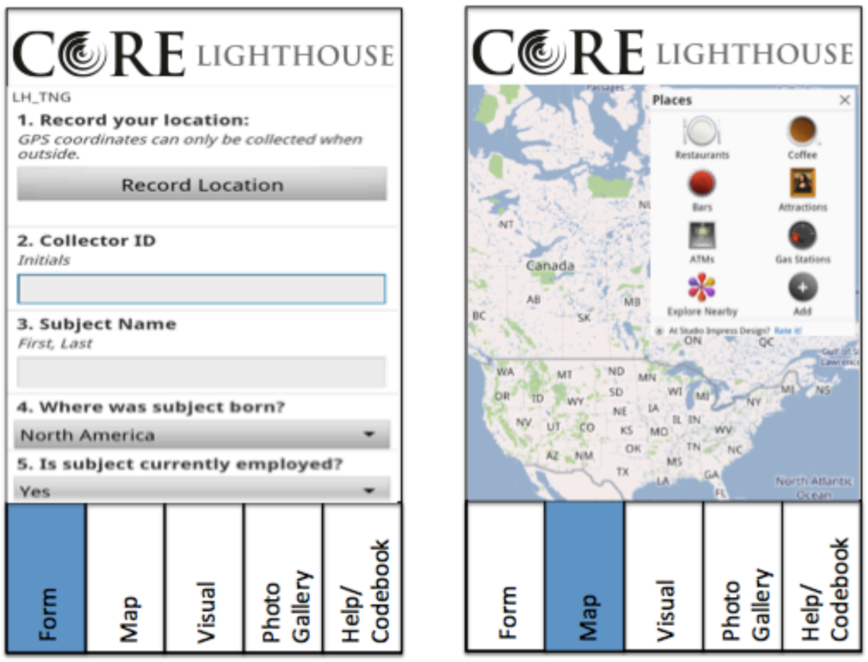
By clicking on the "Form" tab, the application would display the activity that allows users to enter new information collected. The view may be different on a mobile phone, tablet or laptop web browser to take advantage of the larger screen real estate. Figure 7 displays an example of the "Form" and "Map" activities.

2. Map

The "Map" tab would place *drop pins* on various locations on a map that reference different records in the database. Hovering over a *drop pin* would display a summary view of the information from the database that corresponds to that location. Location data could be provided by Google Maps when network connectivity is not an issue, but could prove impossible when there is no access to the Internet. Lack of connectivity could cause delays or other errors when running the application as it

[3] Figures 6, 7, and 8 were created by Mike Stevens of the CORE Lab after discussions with members of the Lighthouse working group. The photo gallery was modified by Marisol Torres to show images from http://vectorcharacters.net.

attempts to update the maps. Leaflet is a free, open-source JavaScript library that could be used to provide interactive maps for mobile devices (Leaflet 2014).


CORE LIGHTHOUSE

LH_TNG

1. Record your location:
GPS coordinates can only be collected when outside.
Record Location

2. Collector ID
Initials

3. Subject Name
First, Last

4. Where was subject born?
North America

5. Is subject currently employed?
Yes

Figure 7. Possible “Form” and “Map” Screens

3. Visual

One of the applications under the umbrella of Lighthouse is a tool that can take data of a specified format, aggregate that data and develop a link diagram. The link diagram is not an analysis of the actual data. Rather, the tool simply scans the data and tallies the number of times a piece of data is referenced and creates a visual representation of the data. The “Visual” tab is one mechanism that agency leaders agree would be needed to incentivize data collection. A collector can visually see what information is lacking in content. Based on their mission, collectors can choose to collect more information about those areas with only a few links, or choose to collect even more information on an area with a large number of links.

Since current operations require Internet access to send the information to a central database to be aggregated, reports and graphical representations of the data must be done externally and downloaded on the mobile device when completed in order for collectors to view these diagrams. The ability for collectors and decision makers to do this locally from any remote location would be a major improvement. Figure 8 depicts example "Visual" and "Photo Gallery" screens. The Java Universal Network/Graph Framework (JUNG) is a framework written in Java that could be used for graphing and visualization on the server side and downloaded to mobile devices. MindFusion, described later, could be used to create the visualizations directly on the mobile device. The Photo Gallery activity would display every photo from the database. Clicking on a photo could provide several options to the user, from displaying information about that person, to creating new links or relationships between people.

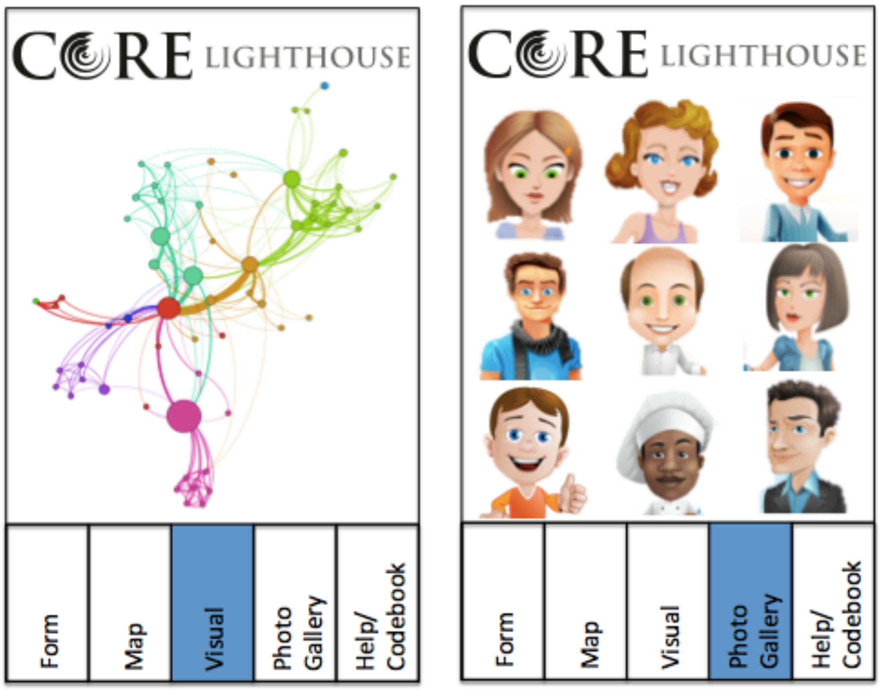


Figure 8. Possible "Visual" and "Photo Gallery" Screens

4. Photo Gallery

The "Photo Gallery" is another capability that the working group users agreed would be extremely beneficial. The user could have several views based on the amount of screen real estate available. For example, on a tablet the screen could display nine or twelve photos while the maximum number of photos that appear on a mobile phone could be four or six. A list view would show three photos with a list of details to the right of the photo. The user could also select a view with one photo that would zoom in on the photo, but also display the rest of the data from the database pertaining to that person. Hovering over a photo could display a summary of the information in the database, while clicking on a photo would take the collector to a "Baseball Card".

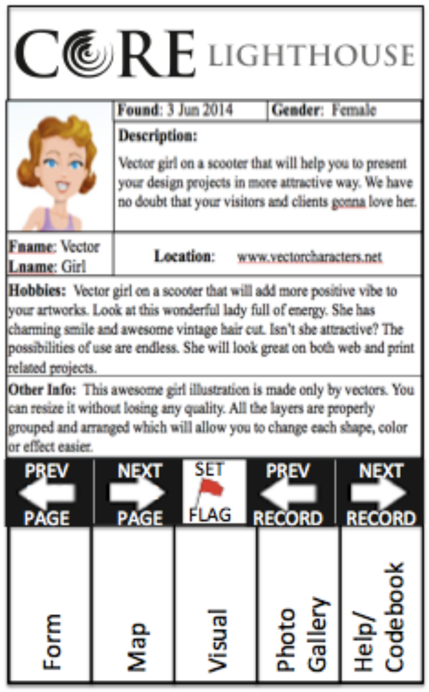
5. Help / Codebook

The "Help / Codebook" screen provides the user the ability to learn more about how the application functions. In addition to explaining useful tips for the data collector, the codebook has definitions and explanations of terms used within the system, as well as the commander's intent of the actual collection activity. Upon clicking on this tab, the collector would be brought to the portion of the codebook that explains information about the last activity. For example, if the user were in the "Photo Gallery", hitting the "Help" tab would display helpful information about the "Photo Gallery". It is a continuous, scrollable view of the entire codebook. Figure 6 displays a sample "Help / Codebook" screen.

6. Other Capabilities

a. Baseball Card

The Baseball Card is a view that collectors could get to from the "Photo Gallery". This screen displays what is referred to as a baseball card because it is basically a summary of data from the database, similar to information that one might find on an actual baseball card of a major league baseball player. The view is a quick view for collectors to familiarize themselves with a specific individual. Double clicking on the photo in the gallery would take the collector to a screen displaying more information from the database about the person selected (similar to Figure 9 below). From that screen, the data collector could modify any of the existing data or simply scroll through the rest of the database information about that person.



CORE LIGHTHOUSE

Found: 3 Jun 2014 **Gender**: Female

Description:

Vector girl on a scooter that will help you to present your design projects in more attractive way. We have no doubt that your visitors and clients gonna love her.

Fname: Vector
Lname: Girl

Location: www.vectorcharacters.net

Hobbies: Vector girl on a scooter that will add more positive vibe to your artworks. Look at this wonderful lady full of energy. She has charming smile and awesome vintage hair cut. Isn't she attractive? The possibilities of use are endless. She will look great on both web and print related projects.

Other Info: This awesome girl illustration is made only by vectors. You can resize it without losing any quality. All the layers are properly grouped and arranged which will allow you to change each shape, color or effect easier.

Figure 9. Possible "Baseball Card" View (after Vectorcharacters 2014)

b. New Links

Working group members agreed that the ability to create new links or relationships between people or other information within the database would be extremely beneficial to Lighthouse collectors. Adding a relationship could be done in several ways, but the most simple would be from the Photo Gallery. For example, a long click on a photo could highlight a photo, placing the user in an "Add relationship" mode. A second long click on a different photo would open another screen where the user would enter information about the type of relationship being added. For example, it could show a screen with the photo or name of person 1 "is a " [kinship type] " "of " photo or name of person 2, where [kinship type] is a dropdown menu from which the collector would select the type of kinship like cousin. If the type of relationship is new, the user could add a new one like " conducts business with" or "purchases goods from" by selecting new from the drop down menu.

c. Export Capability

The "Export" capability is a tool that working group members thought might make integrating with more robust social network analysis tools easier. This tool would enable collectors to create files from database data in specific file formats. For example, the collector could create .dot, .csv, .txt, or .xml files from the data in the local database. They could then send these files to other users in the area via NFC, Bluetooth, or Wi-Fi Direct. Additionally, the files could be sent to remote, external servers when Internet connectivity is established.

d. Database Integration

Another critical piece to make integration seamless is creation of a local database that can actually compare and share information with other local databases. This function would prove extremely powerful. Since the current implementation creates separate tables that do not interact with one another, all information is sent to an external, centralized database for any and all analysis to be performed. Reports are sent back to, or downloaded by collectors when connectivity exists. Smart phones and other mobile devices have processing capabilities that make it possible for them to perform some meaningful analysis locally, rather than using up valuable time and network bandwidth to send the information over the network first.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. METHODOLOGY

The database architecture described in this chapter does not capture information that may be required for the devices to know the authorized users or other authorized devices with which they can communicate. Additional information, perhaps stored in another table, is required for the mobile devices to establish the connections required to keep their data in sync with one another and/or the main database. The specific information required for communication depends on the communication technologies or protocols being used. We discussed some of the considerations for Wi-Fi Direct and Bluetooth information sharing in Chapter II, Sections B and C. We discuss two different methods for sharing data below.

A. UPDATE USING FILE EXPORTS

When synchronizing data between multiple devices, several approaches can be used. To explain the algorithm, we limit the number of devices to four. The algorithm, however, is designed to work with an arbitrarily large number of devices.

SQLite databases on phones are not as robust as normal, full-featured SQL servers. While phones can support multiple tables in an SQLite database, as well as multiple databases on a phone, there are several limitations with mobile versions of SQLite.

One major difference between SQLite and SQL implementations is that an SQLite database can only store three types of data: String, long and double. Normal SQL databases can support many different data types to include Boolean values, binary large objects (BLOBs), variable length character strings (varchars), and several more. To display data types not supported on SQLite, they must be converted properly and then stored locally. They can be reverted back to the appropriate format when being sent to other SQL databases, but that additional conversion may or may not be required. In fact, it may prove to be inefficient to constantly change the file format, but that really just depends on how the external database is being used and what other devices communicate with its database directly. There are APIs that enable developers to do the file conversions with relative ease. For example, there are Java utilities that allow storage of BLOBs in SQLite databases on phones. Those BLOBs can hold photos or other information not normally recognizable by SQLite databases. Since SQLite databases are loosely typed, that is, do not restrict the actual data that is entered into a field, developers are responsible for ensuring the integrity of the database (Murach 2013). This makes it easier to use other data types inside of SQLite databases.

Another limitation of the SQLite databases on a mobile device is direct connectivity to a database. The connection from one mobile device directly to an SQLite database residing on a different mobile device is not permitted. The main reason for this restriction is that external devices would require access to the other devices' file structure, and permissions to access, or modify files on an external device. This capability is intrinsically in violation of basic security protocols. Generally, no one wants external devices to be able to connect to files on local devices, much less modify them. They may want to be able to share a file, or obtain a file from another mobile device, but generally do not want their own files to be modified. Developers have had to come up with creative ways to implement this capability.

With this in mind, there are several routes that application developers can take to make an application seem like it is synchronizing directly. In the method described here, it does not matter which technology is used to connect and send a file. All that matters is that a mobile device must be able to send a file to another device. Basically, any application that attempts to store data into a local database, must also be able to send that data to a file which would be shared with another mobile device. The next steps happen behind the scenes and are not observed by a casual user of the device. The following describes how four mobile devices could perform the synchronization in a background process without regard to the connection technology.

1. Use Case

A team of four data collectors report to a village to obtain information about the local population. There is no existing network infrastructure but the team leader did obtain the download from the master database prior to arriving at the village. He shares the data with the other devices by using a specialized Bluetooth, NFC or Wi-Fi direct application that places the data in a specified location on each mobile device. Again, since complete files can be transferred via each of these methods, the specific connection method does not really matter at this point.

A separate application is opened on each device that parses through the shared file, creates the appropriate local tables and allows the other data collectors to query or modify the stored data. This synchronization method uses a temporary database containing any of the modifications the user makes to the database. If the user never obtained the master file, he or she would still be able to collect new information. The team leader manually sets the device clocks. The application to collect information is started on each device by the individual collectors. Starting that collection application triggers the synchronization application, which runs in the background so it requires no additional action from the information collectors.

All devices in use require uniquely identifiable names and each device has the names of other valid devices stored in local tables. The names could be the MAC addresses of the mobile device but for our example, we will use M1, M2, M3, and M4, respectively. Each device with data to share, or modifications to the database would create a file using a standardized naming convention such as m2_dbname_1.csv. The first part of the name would correspond to the actual machine name of the device that created the file. The second part of the file name would be the database name for which the data belongs and the last part is an index. The index would iterate from 1 to 1000 and repeat. This is merely a precautionary measure to ensure that a device does not overwrite a previously written file that has not yet been aggregated and properly uploaded to the master database. The file could be a .txt, .csv, or .xml file. It really just depends on how the file is being accessed when it is time for synchronization to occur.

This method of synchronization requires that one device is designated as the database master. This is for simplification since true peer-to-peer systems that allow every device to behave as a master can be extremely complex and difficult to implement. This adds several levels of complexity to ensure that data is aggregated properly, duplicated data is not added erroneously and multiple devices do not take multiple attempts to update the same exact data, at the same time. This algorithm avoids those complications by establishing one as the leader or "master". We name M1 as the device that will actually aggregate all data collected and eventually connect to the external database to upload all of the aggregated data.

Each device will store data in a local database as well as create a .csv file in a specified location containing any data that has been modified or added to the database. The new and modified records will have their modified bit set as well as the emp_id and timestamp for either the create_dt or modified_dt. Additionally, any records that should be deleted will have a delete flag set and record the timestamp into the modified_dt and emp_id of the user dictating the change to be made.

When a connection is established, the master device will check the specified location for files with the specific naming convention mentioned earlier. If there are files matching that naming convention, it will check for a semaphore file. The semaphore is an empty file following a similar naming convention except the file extension is .del or .ack extension. The .del informs the device that the file was aggregated and uploaded properly. The .ack informs the local device that a specific file can be deleted. To continue with this example, if m2_dbname_1.del exists, M1 knows that the information has already been aggregated. It deletes the file original .csv file but leaves the .del file as an acknowledgement to M2 that its data were received, aggregated and uploaded to the master database. The next time that M2 connects, if it detects an m2_dbname_1.del file, it knows that those changes were uploaded properly and can remove those records from its temporary database and commit those changes to its local database. M2 would then create an acknowledgement file with the same name and .ack extension. This is a semaphore that informs the local device that other files with the same name can be deleted. The temporary database would still contain all of the new data that was collected between pervious synchronization attempts.

Since we established that M1 would be the local "master" responsible for aggregating data collected in the field, the other devices would not pull new data unless M1 specified that new data exists. So, if no device has changes to be made, no files will be created or shared. However, if there were a change, M1 would inform its peers that a database update exists after a specified amount of time or drop the most current version of data into the appropriate local location at the touch of a button. When those devices want to obtain updated data, they would only obtain actual updates from M1. Although they would still obtain copies of every device’s .csv, .ack and .del files, they would not actually use any of the data in those files. Those files are simply replicated to ensure that M1 could obtain M3’s update from M2 or M4 in the even that M3 could not establish a connection to M1 directly. This method prevents loss of data in the event that network connectivity is nonexistent.

This method is not very different than the current implementation of Lighthouse in that updated information is gathered and used to generate reports that are stored on the mobile device. These files are stored in an “outbox” for use when connectivity is not an issue. So, when there exists a connection to the Internet, the “outbox” transmits the messages that have not been sent.

This is not truly a departure from current operations. The only benefit of this capability would be that users could share their information with one another so that in the event that one of their devices is compromised or lost, the information collected on that device is not lost. Rather, it is on another, local mobile device that can send the files to the cloud when connectivity is restored. While useful, this extension has limited benefit.

It is more useful to provide this capability as an important adjustment is made to the application's current capability. The current application must include a reader that can parse through the files sent by other users and stores the information in each device's own local database. Additionally, the new generation would have to support additional capabilities such as visualization and photo gallery search to be truly useful. This would mean the collectors could potentially benefit from the information that was collected by others. Absent these, the data would still need to be uploaded to the cloud for exploitation.

B. DATABASE SYNCHRONIZATION

If we want the information collected by users to be shared between collectors more readily so that the information could be used by the collectors on the ground, the previously described implementation would not be enough.

In the following approach, instead of storing and sending actual files, we share individual objects or records. In order for this approach to work, there are several key requirements that need to be satisfied.

The first requirement is that the system uses one of the three communications mechanisms described in Chapter III, Section B2. Each of these connection mechanisms has specific requirements in order to share data. Bluetooth requires MAC addresses while Wi-Fi Direct requires IP address. These details are not critical for this research since IP addresses can be assigned by group leaders upon creation of a group. At a minimum, all devices must know the device name and MAC address of all devices to which it must send information. Although this is not a requirement when using Bluetooth technology, as mobile devices truly need not know to whom they wish to connect in advance. This is a requirement since developers and users wish to limit information leaks to unauthorized devices. This acts as an access list, storing specific information that would make connecting to a device more rapid. Other information would be stored to ensure synchronization functions are executed properly.

1. Data Sharing Algorithm

The algorithm below describes an implementation that would work for sharing data. It would ensure that new and modified data is only sent to other devices when certain conditions exist, preventing unnecessary traffic. Additionally, it prevents against out of sync databases by attempting to send updates at least a specific number of times (three in our case) unless it receives an acknowledgement that data was successfully received.

When a device has data to send, it starts counting from zero. It uses a sent and ack array to keep track of which devices have yet to receive the data. If the sent flag is 0, the device will attempt to send the data and set the sent flag to one after the data has been successfully sent. If the sent flag is 1, it moves on to the next device awaiting the data. After going through each device in the device list, it resets the sent array, increments the number of attempts and starts going through the device list again. After the third attempt to send, no further attempts to send that data are made. Pseudo code for the algorithm is provided below.

Each device will perform the following steps to send:

```
Step 1. Create devList = DL[ ] - DL[ i ] ; local_device = DL[ i ];
Step 2. Initialize arrays and variables:
            n= | DL[ ] |;
            sent_i[ ] = 0;
            ack_i[ ] = 0;
            rec_i[ ] = 0;
            attempts = 0;
            maxTimes = 3;
Step 3. Gather data to send:
         DATA = getData( ) {
            check messages table {
                if old messages still unsent, re-send
            }
            results = query( ) ; //select * from table where modified == "Y";
            if (results == null or " " ) {
                EXIT // do not continue; wait until there is something to send
            }
             else {
                mId = getMessageID( ); // get next available message id
                    return newData = DL[ i ] + ", " mId + ": " results ;
            }
         } // end of getData( )
Step 4.  While ( attempts < maxTimes ) {
            for ( i in devList ) {
                if ( ack[ i ] != 1 ) {
                    if ( sent[ i ] == 0 ) {
                        sendData( devList[ i ], DATA ) {
                            **here, i is the destination.
                            **DATA contains the source, mId and data updates
                        };
                    sent[ i ] = 1;
                    } // end sent[ i ] check
                } // end if ack[ i ] check
                next i;
            } // end for-dev loop
            attempts++;
} // end while
```

2. Receiving Data Algorithm

The algorithm below describes an implementation that would be used when receiving data from an authorized device. It would store the data received in its local SQLite database and respond with an acknowledgement to the sender. The sent and ack arrays maintain the statuses of the sent and ack flags respectively and are the same arrays mentioned in the sharing algorithm. The acknowledgement is an effort to prevent unnecessary re-transmission of data. When a device has received data from an external device, the device sends an ack containing the message ID to the sender of that message. Upon receipt of an ack, the message sender sets the ack flag and no longer attempts to send that message to the device for which the ack was received. Pseudo code for the receiving algorithm is provided below.

Each device will perform the following steps when receiving data:

```
Recall from earlier that $D_i$[ ] = DL[ ] - DL[ $i$ ]
Step 1. RECEIVED = recData( ) {
            source = DATA[ 0 ]; m_Id = DATA[ 1 ]; newData = DATA[ 3 ];
            determine message type
                if type == ack ,
                    return ack
            else newData == (string of records;
                                 comma separated columns;
                                 records separated by \n)
        return newData;
        }; // What was received -- (ack or newData )
Step 2.   $idx$ = getIndex( ); // get position in array for device that sent the data
Step 3. switch ( RECEIVED )
        {
        case “ack” :
            $ack_{idx}$[ devList[ $idx$] ] = 1
            break;
        default :
            updateLocalDB() {
            update = sql code that enters information received into local database
                if record already exists, check create_date and modified_date
                    save the newer record;
                if questionable, save to error table.
                set success to true if update successful, else success = false
            };
            if ( success ) {
                sendAck( devList[idx], m_Id) {
                    send from my_device to devList[idx]
                    ---  dev, mId stored
                };
            }
        };
```

3. Algorithm Walk-Thru

As one reviews the algorithm, one should notice that it does not specify a specific action that triggers or starts the sharing. This was done intentionally. The trigger could be based on time, a certain number of changes or new records added to the local database, a button pressed by the user, or other action specified by the developers or decision makers. Additionally, this research focused on an algorithm that would be as generic as possible so the method of connectivity (Wi-Fi Direct, Bluetooth, WAN, LAN, etc.) was not important either.

As an example, the data sharing and receiving programs can be separate activities triggered by loading a data collection form. These activities can be separate applications, or part of the data collection application. In this mode, these would be running in the background requiring no action from the data collector.

As mentioned previously, after a certain amount of time, or after collecting a certain amount of records, the sharing program verifies that there is data that to be shared. It tracks whether data needs to be sent by checking the sent and ack arrays. This is not the only way this algorithm could work. It could also use a separate message table that holds the message IDs of all messages sent and received. Rather than checking the position in the sent and ack arrays, it would check the field within the message table corresponding to the device. If the ack for a message ID for a specific device is not received, it retransmits that message to the specific device. In either case, this is what occurs in the Prep DATA portion of the flowchart. It will start the attempt count and send data if an ack has not yet been received. Upon sending data, it will set the sent flag corresponding to destination device. Otherwise, it will move on to the next device in the device list until it has gone through all of the devices in the device list.

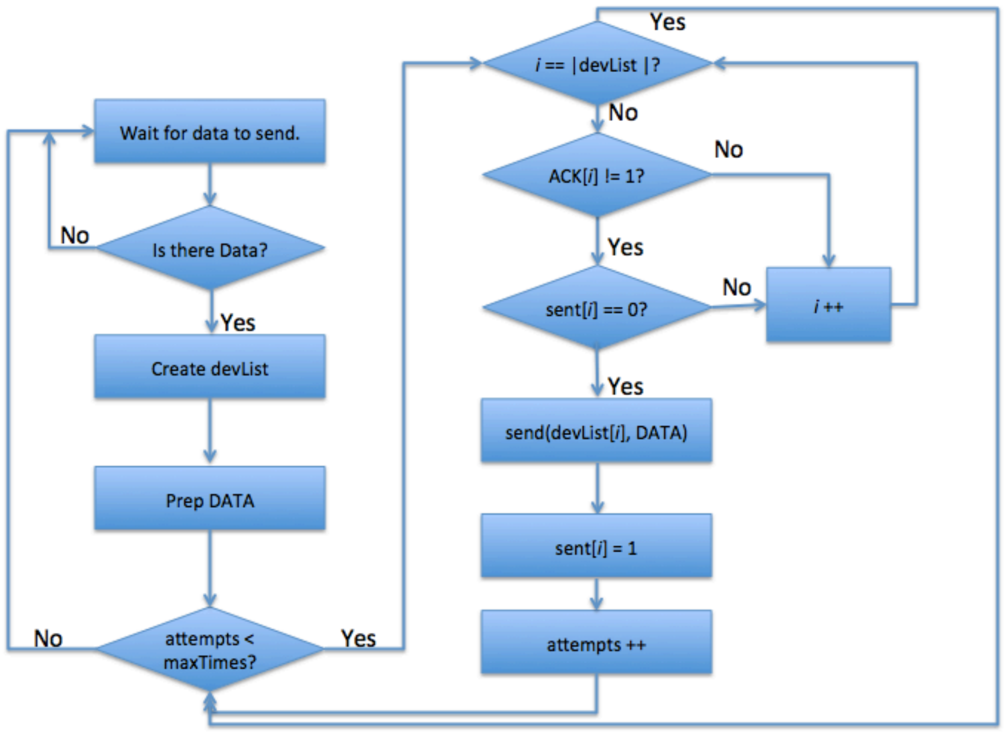


Figure 10. Sharing Data Flowchart

The receiving flowchart below depicts the actions taken by the receive listener. The listener will begin running on the mobile device upon start up of the collection form. As in the previous explanation, there is no special action required of the user to start the listener. This program is running in the background unbeknownst to the user and does not inhibit the users' ability to enter new data in any way.

The device is listening for messages from any of the devices in its device list array. When it receives a message, it checks the source of the message. The sharing program uses this information to identify the index in the array that will be modified. Then, it starts to parse the message received to verify if the message is actual data or an acknowledgement. If it is an acknowledgement, the index for the source device in the ack array is set to 1 and the device goes back to listening for data.

If the message is an actual message, it is parsed and the local database is updated.

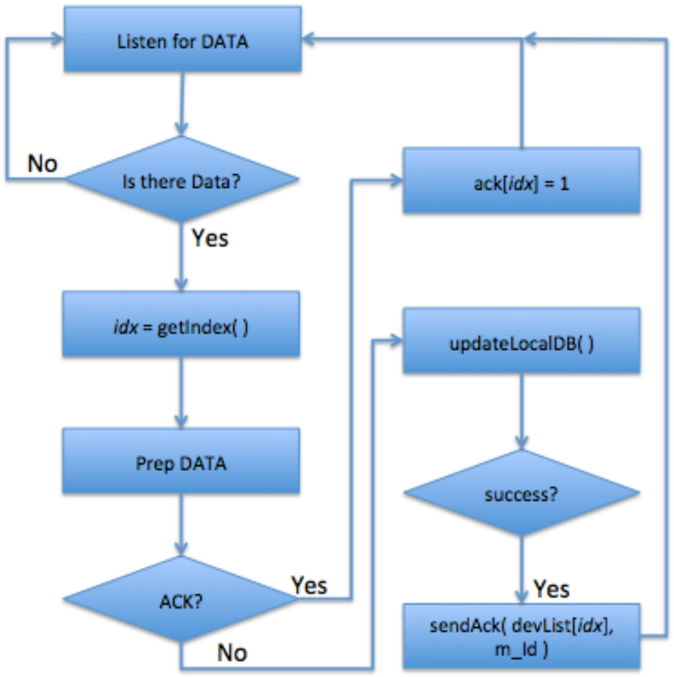


Figure 11. Receiving Data Flowchart

4. Implementation

The prototype developed for this demonstration was created using a free Android Open Source Project Bluetooth Chat Application found on GitHub.[4] The code was modified using the Android Developer Tools Build: v22.6.2-1085508, which includes the Eclipse Platform. The application was tested on Android devices with API Level 15 (Ice Cream Sandwich) or higher. This was to ensure stability and consistency of the application while using the local SQLite database and Bluetooth technology simultaneously, as well as ensuring compatibility within Eclipse.

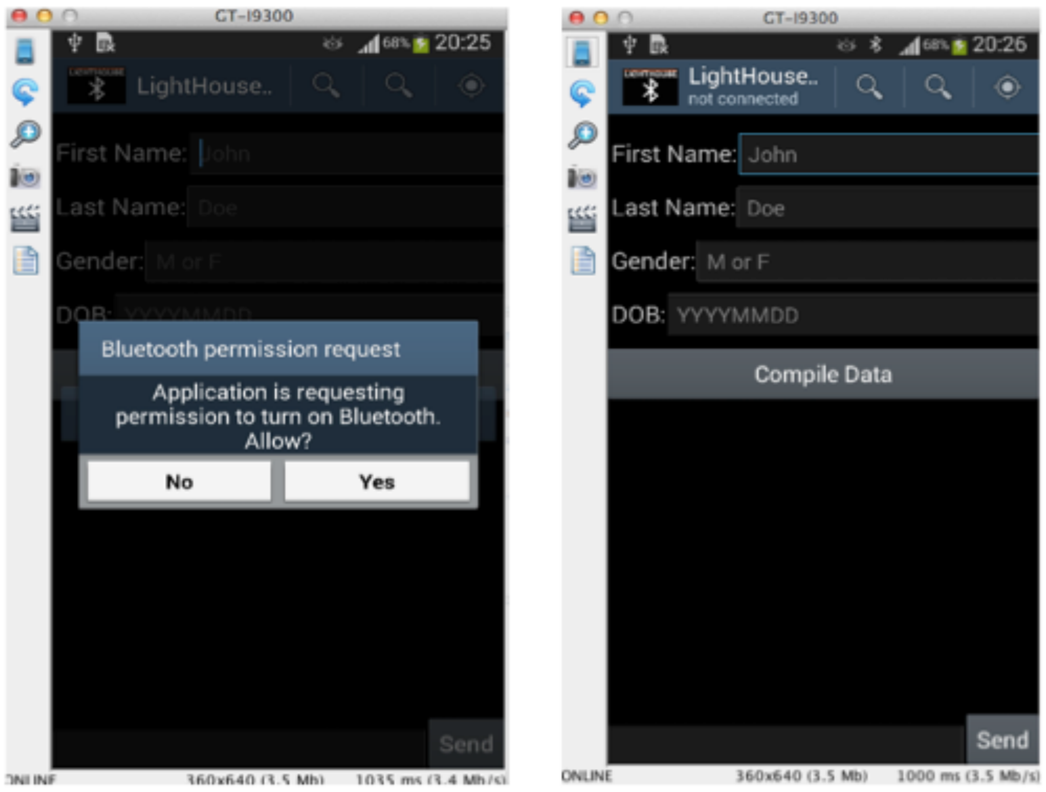
The discussion below walks through the above algorithm and shows three devices. Table 1 lists the devices used, their device names, operating systems and database version.

[4] According to GitHub.com, their website was founded by Tom Preston-Werner, Chris Wanstrath, and PJ Heyett and is one of the most popular code hosting/sharing websites in use.

DEVICE #	DEVICE NAME	TYPE	DESC	Database
1	NPS5	GT-19300	Android v4.03	SQLite
2	NPS1	Nexus 7	Android v4.03	SQLite
3	Galaxy Nexus	Galaxy Nexus	Android v4.03	SQLite

Table 1. List of Devices Tested

The application starts off by checking whether Bluetooth is on or off. If it is off, the user is prompted to turn on Bluetooth as depicted by the Program Action Sequence in Figures 12, 13 and 14.



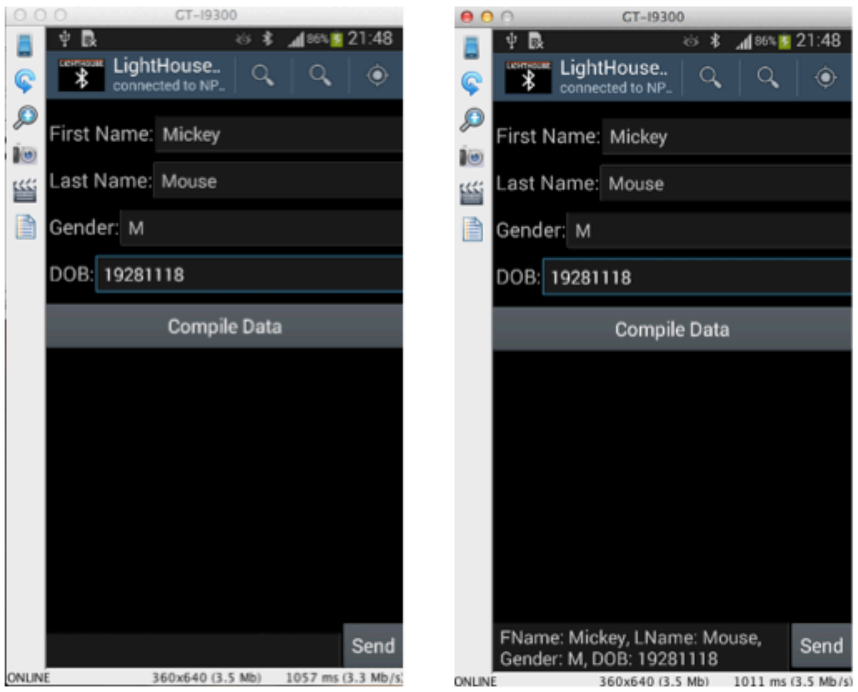
Bluetooth permission request

Application is requesting permission to turn on Bluetooth. Allow?

Figure 12. Start of Program Action Sequence of NPS5

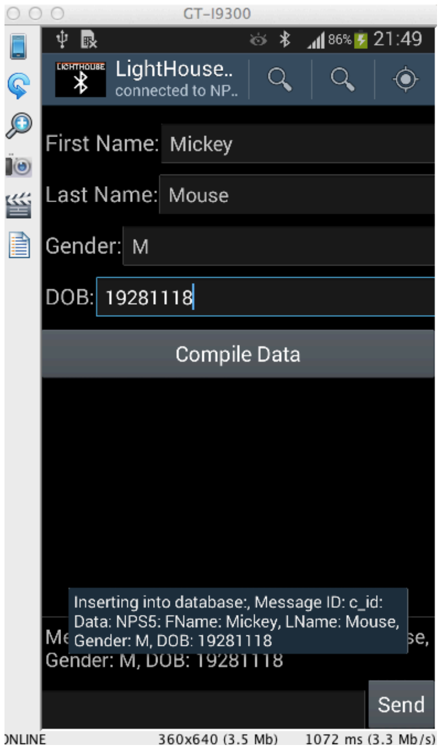
If Bluetooth is on, it opens the data entry form with example text so the user knows the correct format of each field. This application forces the user to scan for devices manually, however, this capability can be programmed to execute in the background. The intent is for the device to make connections automatically in the background without requiring any action by the collector. In this way, the collector would simply continue to collect data, without having the additional burden of sharing the information, or verifying that data is being shared. This application forces the actions to be done manually in order to demonstrate the different actions that take place to establish a connection with nearby data collectors.

The prototype discussed here does not trigger the algorithm described. Rather, it simulates similar actions. For example, it forces the user to manually connect to an external device prior to entering any records into the database. In actuality, the connections need not be established upon startup of the application. This was a design choice to maintain the chat functionality. The interface allows the user to enter one record. When the user hits the compile button, the data is prepared for sending. The user reviews the information being sent at the bottom of the screen and must hit the send button to send the data. If for any reason the connection to the external device is lost, the user must re-establish connectivity prior to sending any data. On the receiving end, the received message is stored in the device's local database. All of the required actions occur manually in this prototype. The actual application would run behind the scenes.


First Name: Mickey
Last Name: Mouse
Gender: M
DOB: 19281118

FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118

Figure 13. Steps 3 and 4 of NPS5



First Name: Mickey

Last Name: Mouse

Gender: M

DOB: 19281118

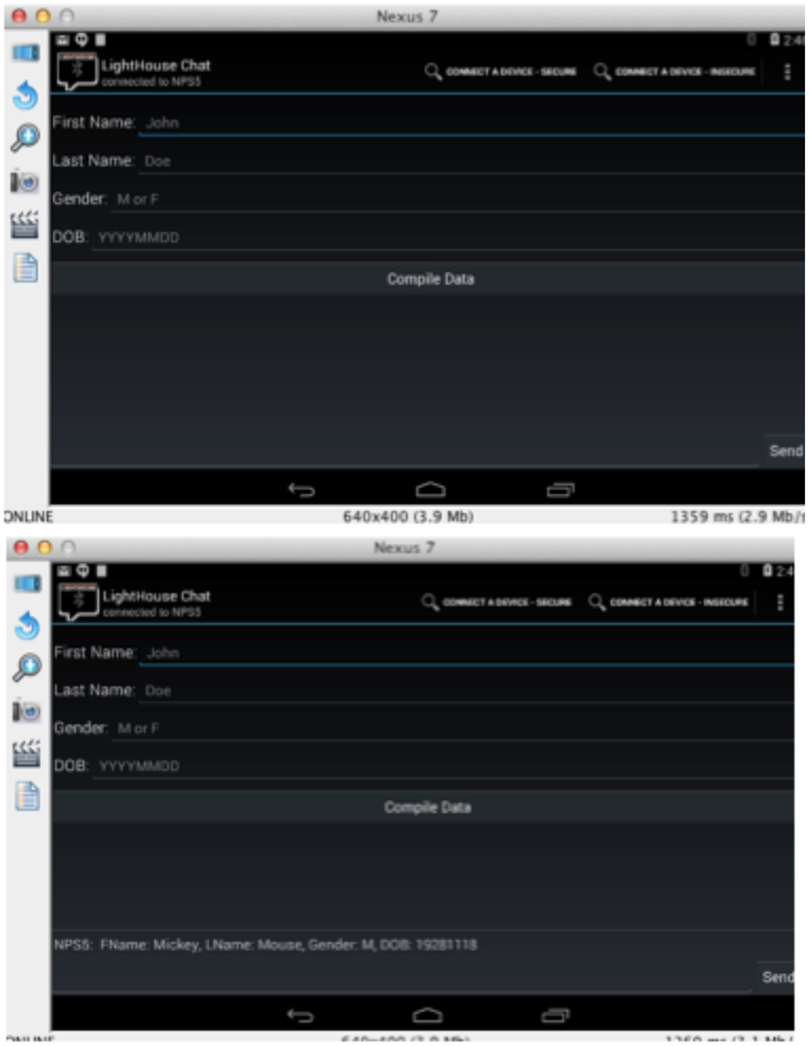
Inserting into database:, Message ID: c_id: Data: NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118

Me … se,
Gender: M, DOB: 19281118

Figure 14. Final Step of NPS5

Stepping through the program action sequence above, we see the various screens seen by a user, NPS5, who is already connected to an external device, NPS1. Data is entered into the collection form, compiled, displayed at the bottom of the screen and sent to the external device. After hitting the send button, the information is entered into the local SQLite database and sent to the external device.

Figure 15 and 16 shows the screens viewed by NPS1. Starting from the top left, we see that this tablet is connected to an external device named NPS5. The screen on the top right shows that it received data from NPS5 and displays the information received. There is no photograph of the message stating that the information has been stored onto this device's local SQLite database because even with a long toast message, I was not able to perform the screen capture quickly enough.

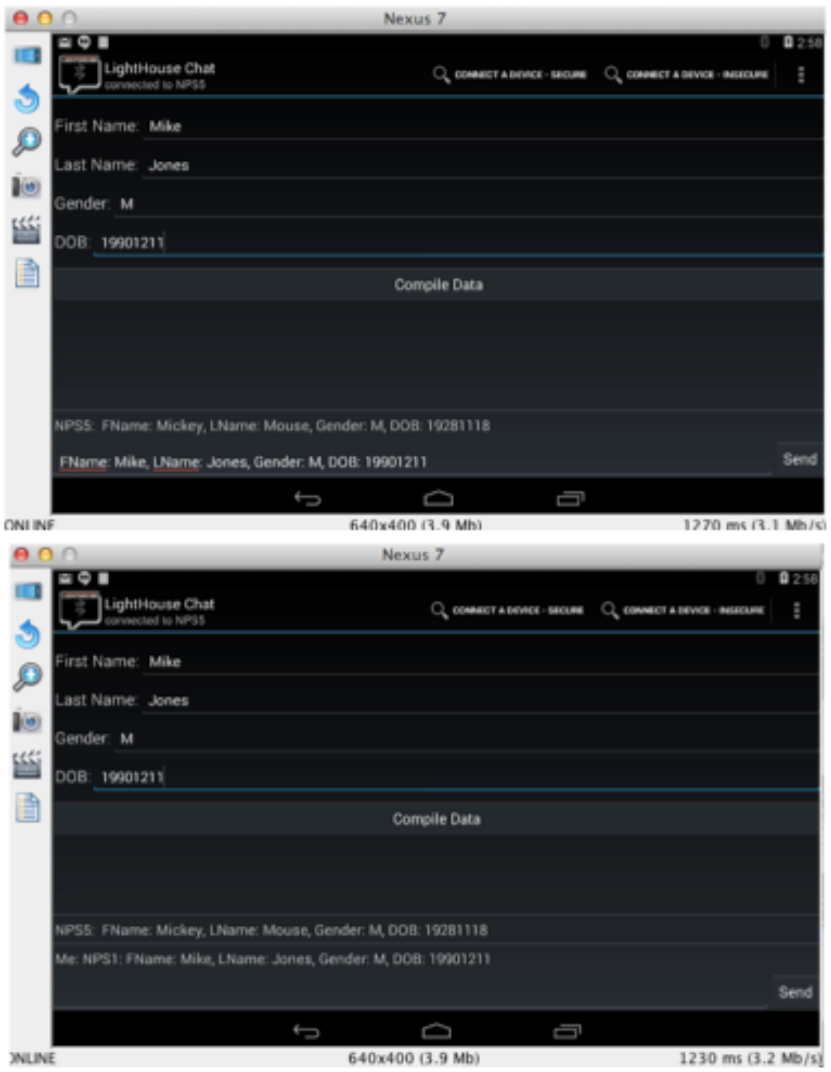


First Name: John
Last Name: Doe
Gender: M or F
DOB: YYYYMMDD

NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118

Figure 15. Start of Program Action Sequence for NPS1

In order to demonstrate that data is sent in both directions, next, NPS1 enters information on the top of Figure 16 and reviews it on the bottom of the tablet after hitting the "Compile Data" button. Upon hitting the send button, we see the updated screen with both records displayed in the photo to the bottom of Figure 16.



First Name: Mike
Last Name: Jones
Gender: M
DOB: 19901211

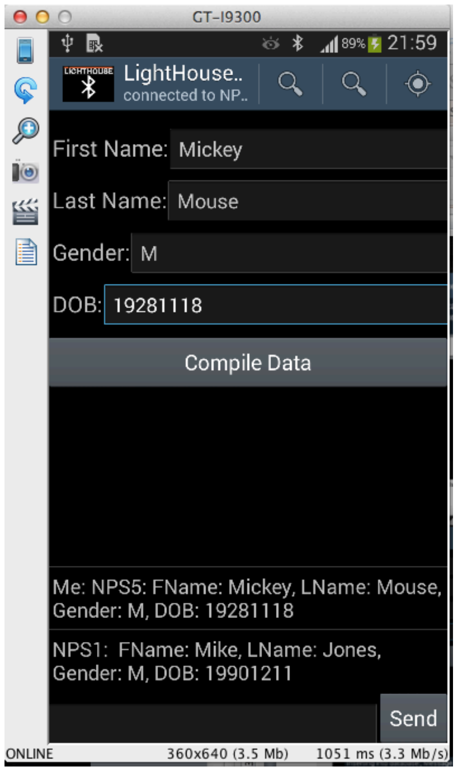
NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118
FName: Mike, LName: Jones, Gender: M, DOB: 19901211

First Name: Mike
Last Name: Jones
Gender: M
DOB: 19901211

NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118
Me: NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211

Figure 16. Screens 3 and 4 of NPS1

Figure 17 shows the event as seen by the NPS5 device.



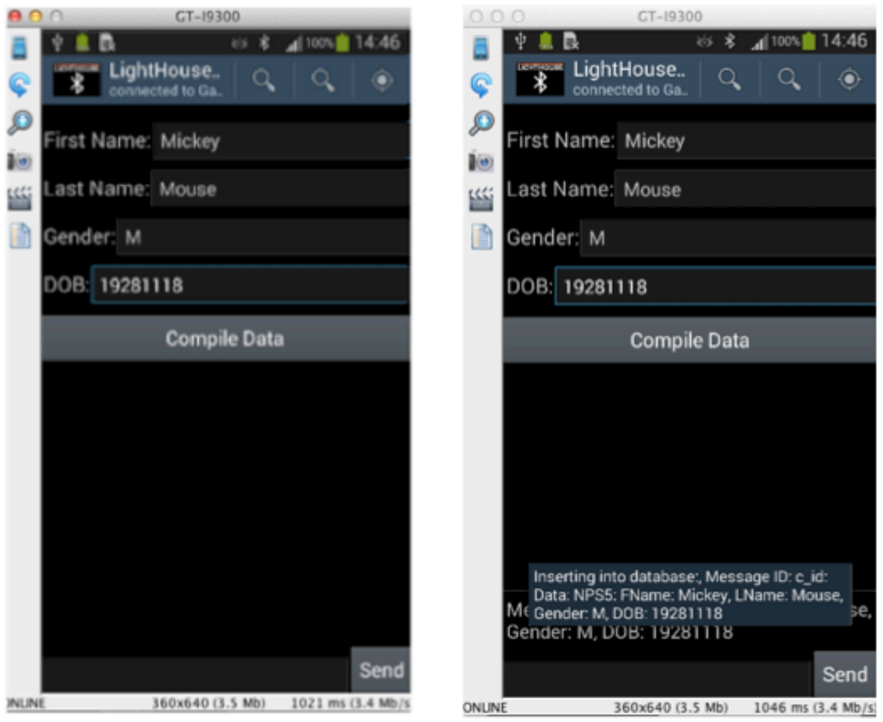
First Name: Mickey
Last Name: Mouse
Gender: M
DOB: 19281118

Me: NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118

NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211

Figure 17. NPS5 screen after receiving data from NPS1

After the NPS5 has finished sharing its information with NPS1, it checks its device list to see if there are other devices that still require the information. The device executes the algorithm once again to share with a third device called Galaxy Nexus. Figures 18 through 22 show the program sequence for NPS5 sharing its data with the third device, Galaxy Nexus. Again, although the prototype shares one record at a time, the actual application would execute a query to obtain all of the updated or new records and send all the records to the next device at the same time. The receiving device would parse through the message received and update its local database.

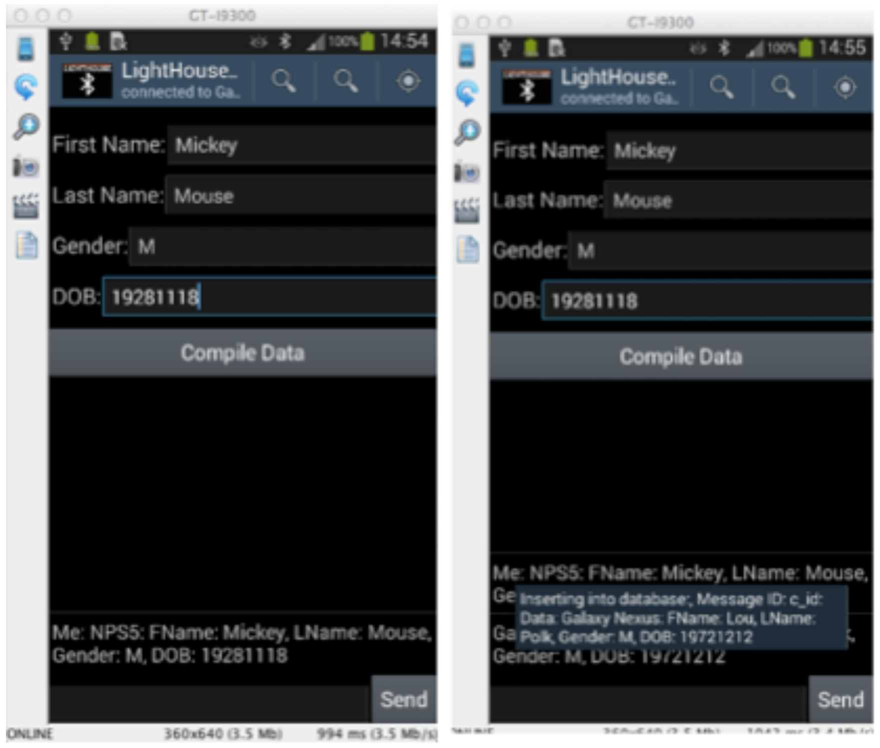


First Name: Mickey
Last Name: Mouse
Gender: M
DOB: 19281118

Inserting into database:, Message ID: c_id: Data: NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118

Gender: M, DOB: 19281118

Figure 18. NPS5 sharing data with Galaxy Nexus



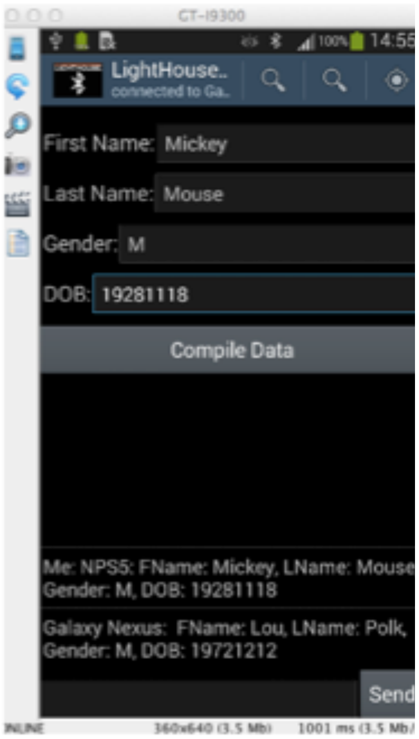
First Name: Mickey
Last Name: Mouse
Gender: M
DOB: 19281118

Me: NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118

Inserting into database:, Message ID: c_id: Data: Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

Gender: M, DOB: 19721212

Figure 19. NPS5 receiving data from Galaxy Nexus

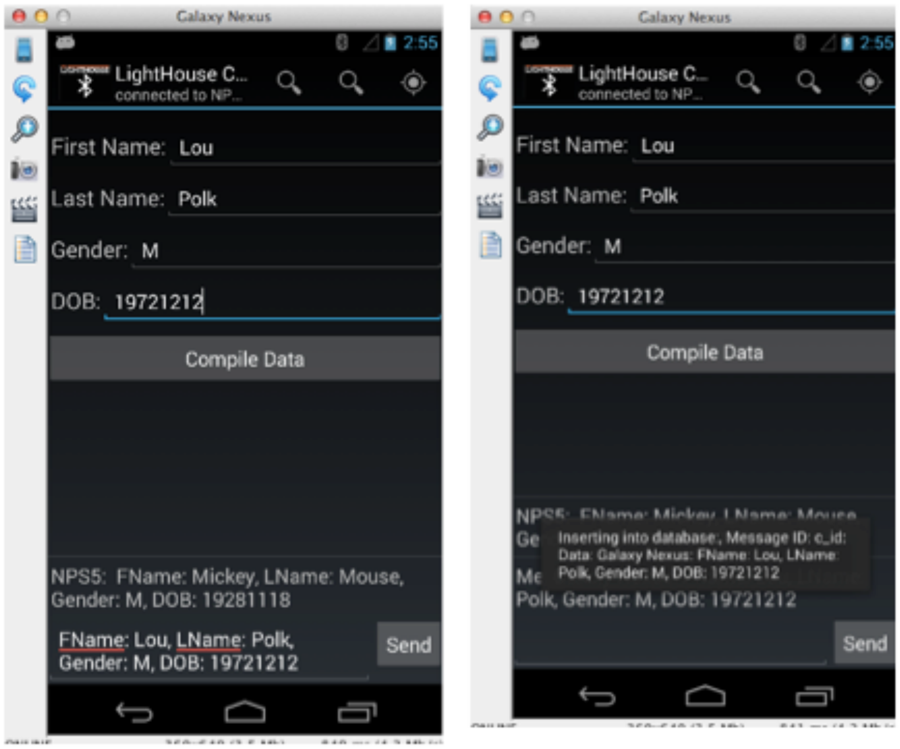


First Name: Mickey
Last Name: Mouse
Gender: M
DOB: 19281118

Me: NPS5: FName: Mickey, LName: Mouse
Gender: M, DOB: 19281118

Galaxy Nexus: FName: Lou, LName: Polk,
Gender: M, DOB: 19721212

Figure 20. NPS5 after inserting data from Galaxy Nexus



First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

NPS5: FName: Mickey, LName: Mouse,
Gender: M, DOB: 19281118

FName: Lou, LName: Polk,
Gender: M, DOB: 19721212

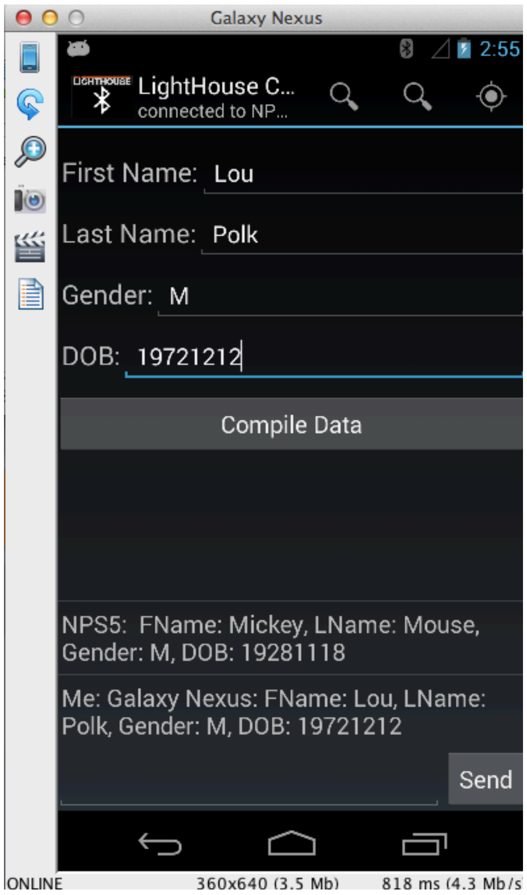
First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

NPS5: FName: Mickey, LName: Mouse
Ge

Inserting into database:, Message ID: c_id:
Data: Galaxy Nexus: FName: Lou, LName:
Polk, Gender: M, DOB: 19721212

Me
Polk, Gender: M, DOB: 19721212

Figure 21. Galaxy Nexus sharing data with NPS5

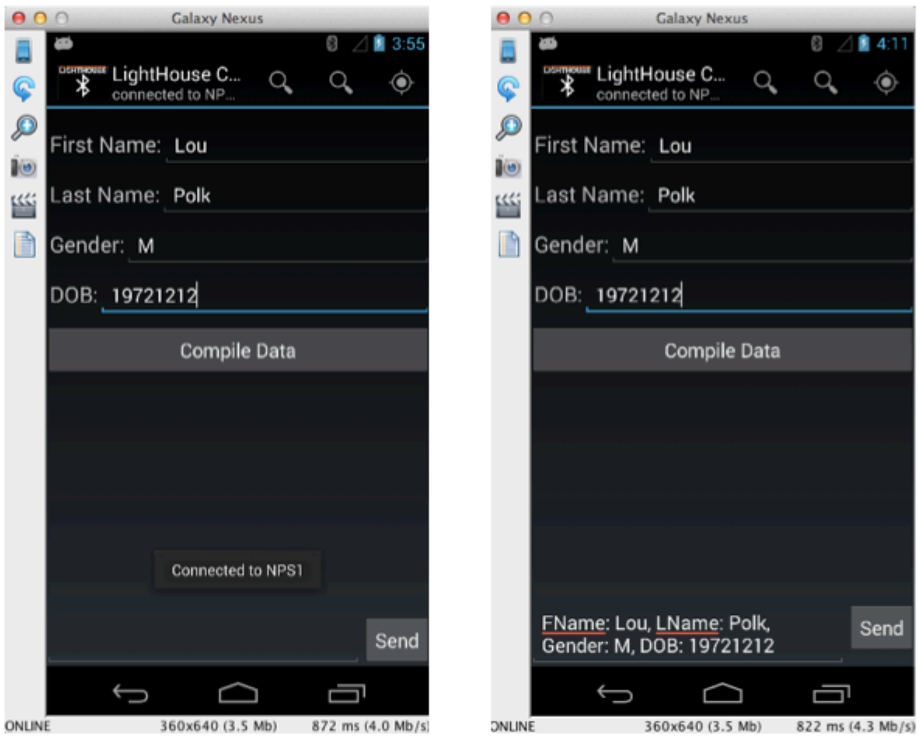


First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118
Me: Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

Figure 22. Galaxy Nexus after sharing data with NPS5

At this point, NPS5 has shared all of its information with the other two devices that required its updates. Next, the second and third devices would make the required connections to share their respective data. Figures 23 through 25 depict the actions taken by NPS1 and Galaxy Nexus in order to share their data from the Galaxy Nexus perspective. Figures 26 through 28 show the same actions from the NPS1 perspective.



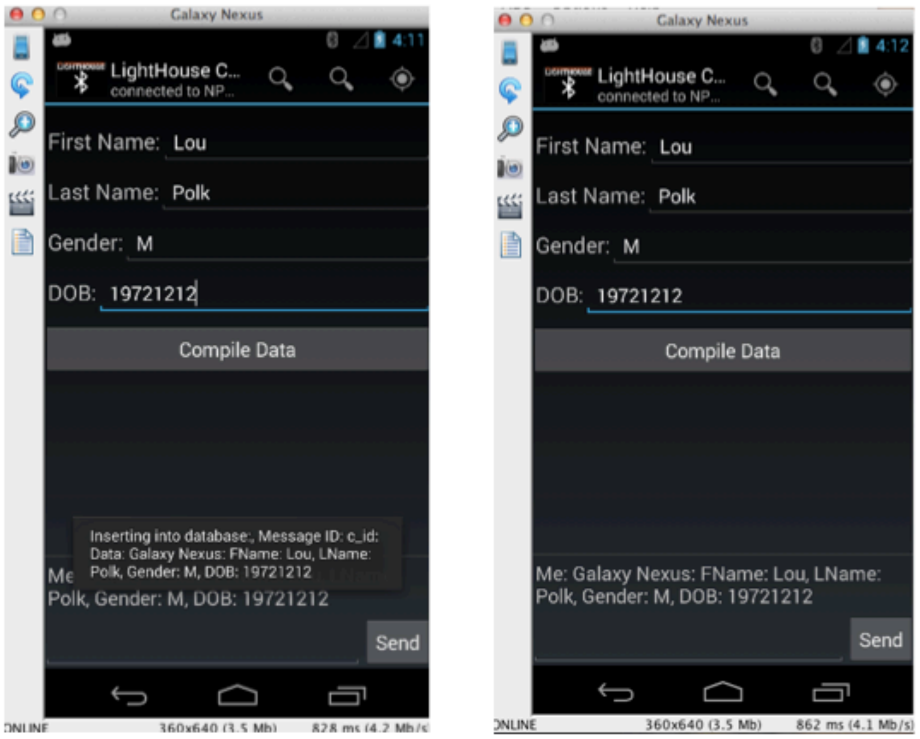
First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

Connected to NPS1

First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

FName: Lou, LName: Polk,
Gender: M, DOB: 19721212

Figure 23. Galaxy Nexus sharing data with NPS1



First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

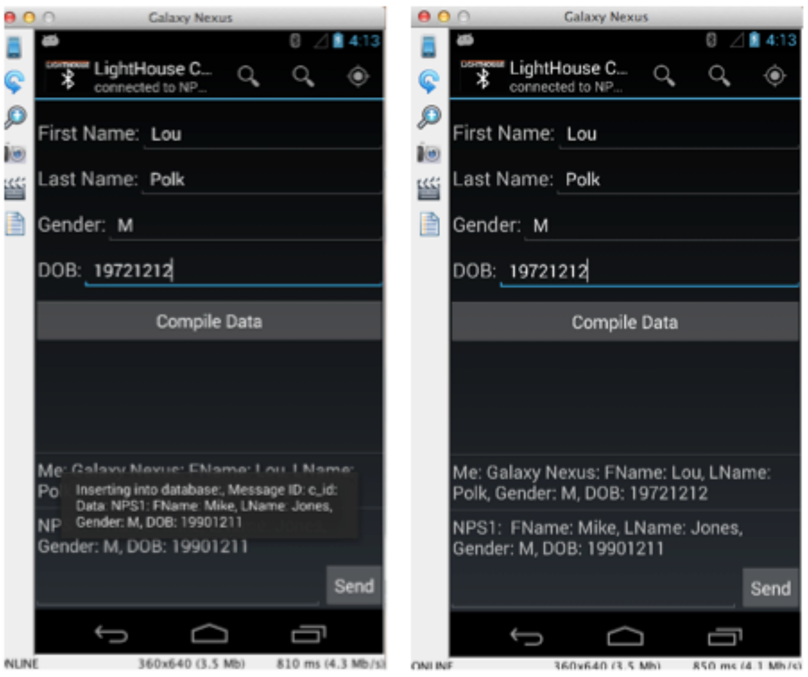
Inserting into database:, Message ID: c_id:
Data: Galaxy Nexus: FName: Lou, LName:
Polk, Gender: M, DOB: 19721212

Me
Polk, Gender: M, DOB: 19721212

First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

Me: Galaxy Nexus: FName: Lou, LName:
Polk, Gender: M, DOB: 19721212

Figure 24. Galaxy Nexus after sending data to NPS1



Galaxy Nexus

LightHouse C...
connected to NP...

First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

Inserting into database:, Message ID: c_id: Data: NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211

Gender: M, DOB: 19901211

Galaxy Nexus

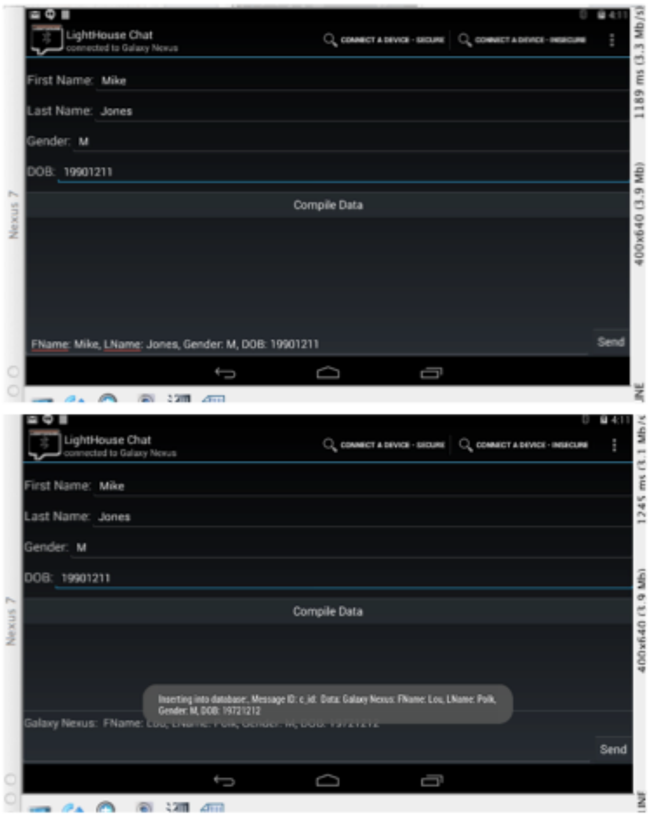
LightHouse C...
connected to NP...

First Name: Lou
Last Name: Polk
Gender: M
DOB: 19721212

Me: Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211

Figure 25. Galaxy Nexus after receiving data from NPS1



LightHouse Chat
connected to Galaxy Nexus

First Name: Mike
Last Name: Jones
Gender: M
DOB: 19901211

FName: Mike, LName: Jones, Gender: M, DOB: 19901211

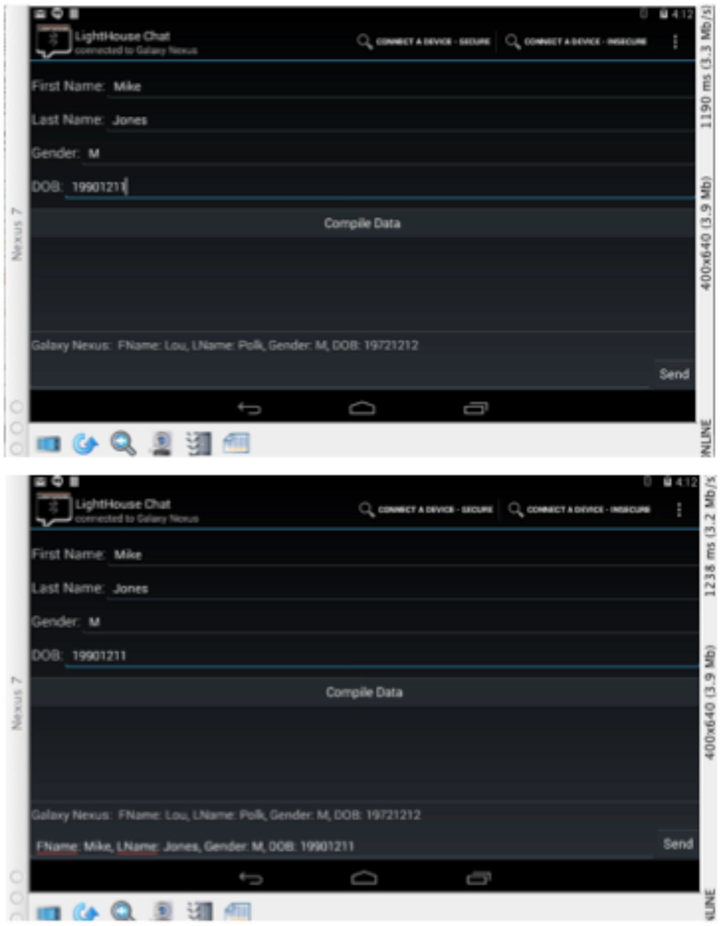
LightHouse Chat
connected to Galaxy Nexus

First Name: Mike
Last Name: Jones
Gender: M
DOB: 19901211

Inserting into database:, Message ID: c_id: Data: Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

Figure 26. NPS1 receiving data from Galaxy Nexus



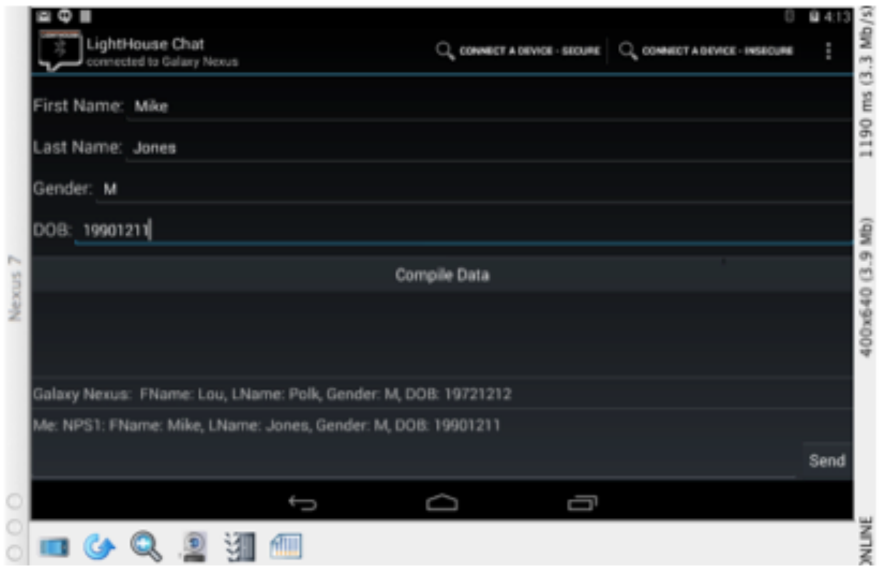
First Name: Mike
Last Name: Jones
Gender: M
DOB: 19901211

Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

First Name: Mike
Last Name: Jones
Gender: M
DOB: 19901211

Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212
FName: Mike, LName: Jones, Gender: M, DOB: 19901211

Figure 27. NPS1 sending data to Galaxy Nexus

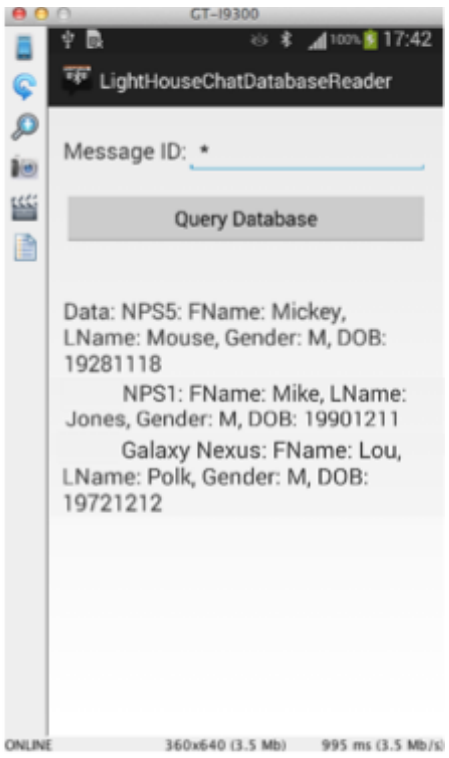


First Name: Mike
Last Name: Jones
Gender: M
DOB: 19901211

Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212
Me: NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211

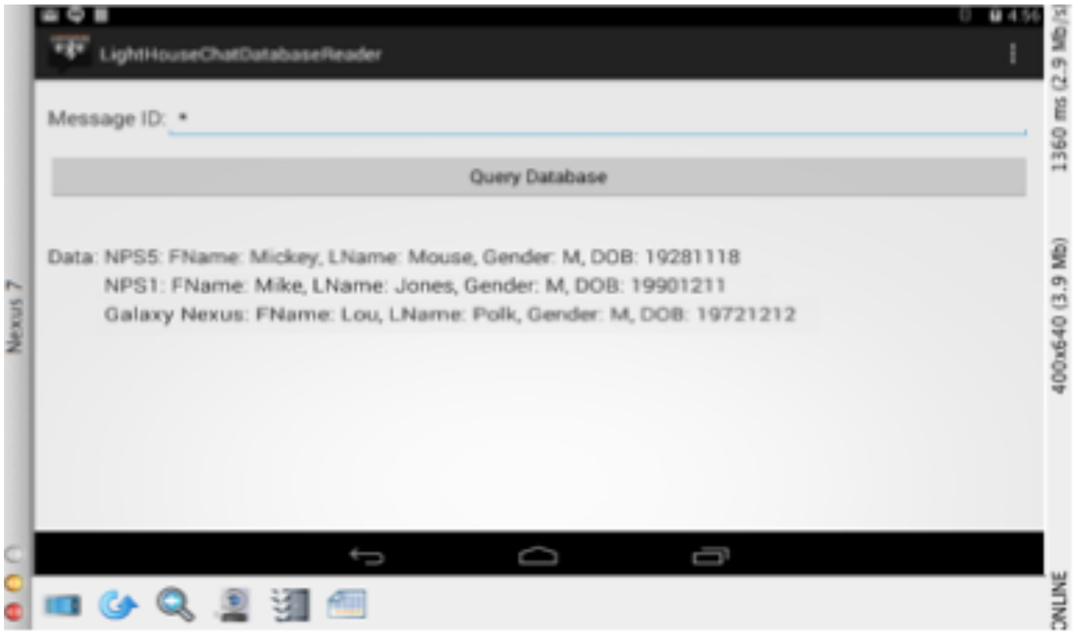
Figure 28. NPS1 after sharing data with Galaxy Nexus

After this iteration of the algorithm, the databases on all three devices have all three records that needed to be shared. To show that all the records are on each device, a separate database reader was used to display the records of the database. Figures 29, 30, and 31 show the entries in each local database.



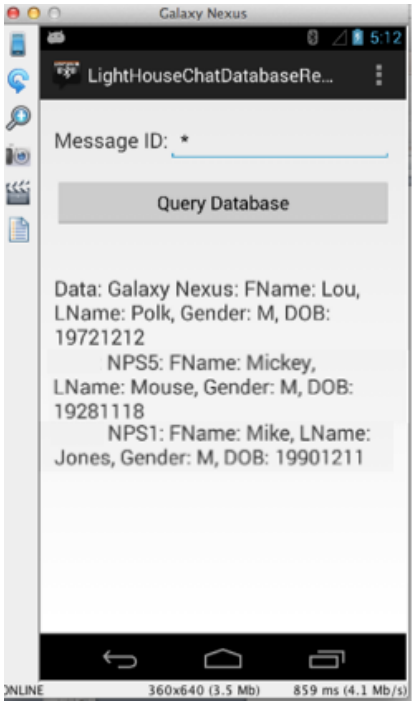
Data: NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118
NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211
Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

Figure 29. NPS5 Database Records



Data: NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118
NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211
Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212

Figure 30. NPS1 Database Records

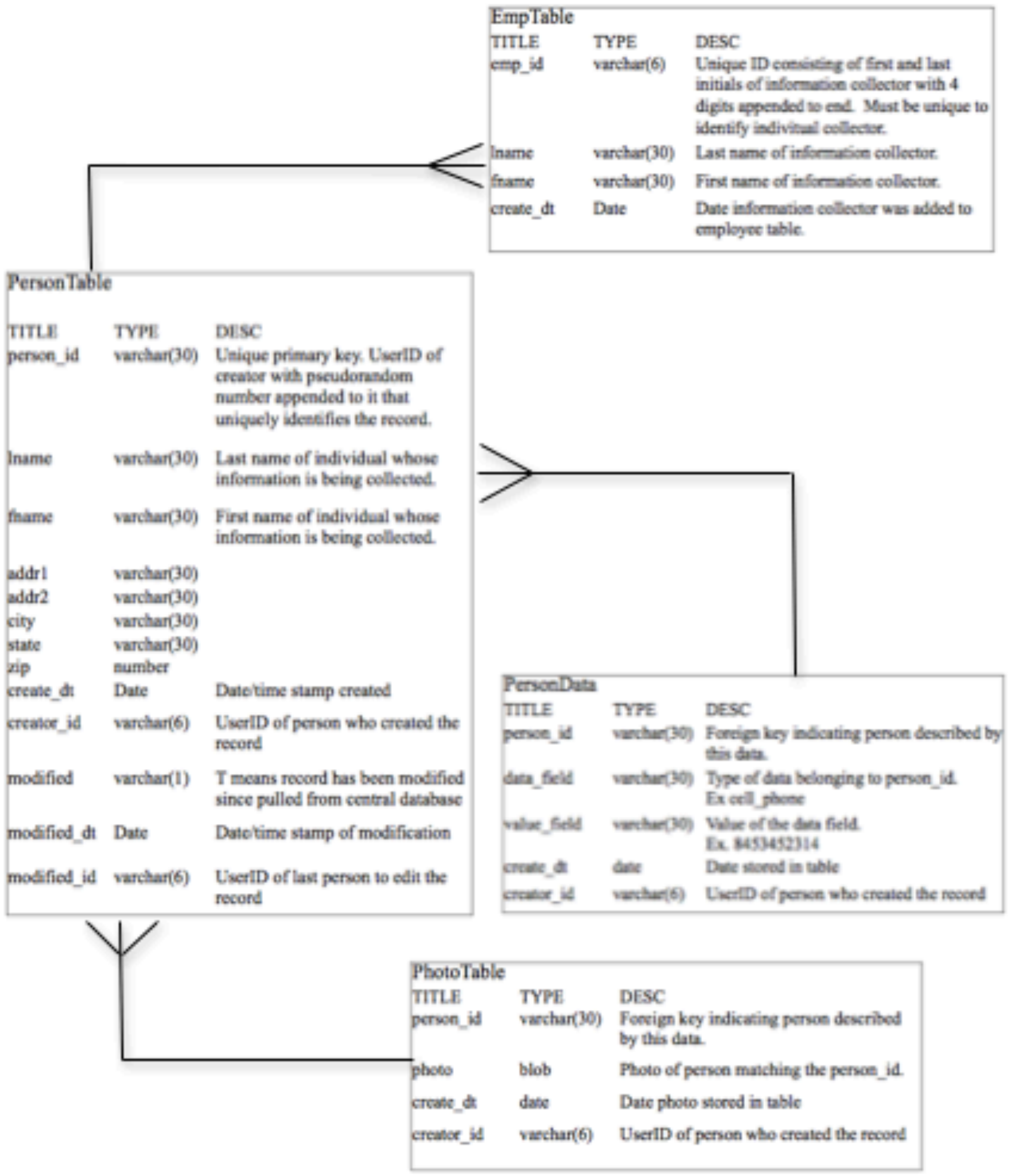


Data: Galaxy Nexus: FName: Lou, LName: Polk, Gender: M, DOB: 19721212
NPS5: FName: Mickey, LName: Mouse, Gender: M, DOB: 19281118
NPS1: FName: Mike, LName: Jones, Gender: M, DOB: 19901211

Figure 31. Galaxy Nexus Database Records

C. DATABASE STRUCTURE

Figure 32 depicts a sample database structure for a centralized database. Its format is slightly different from the SQLite database on a mobile device.



EmpTable

TITLE	TYPE	DESC
emp_id	varchar(6)	Unique ID consisting of first and last initials of information collector with 4 digits appended to end. Must be unique to identify indivitual collector.
lname	varchar(30)	Last name of information collector.
fname	varchar(30)	First name of information collector.
create_dt	Date	Date information collector was added to employee table.

PersonTable

TITLE	TYPE	DESC
person_id	varchar(30)	Unique primary key. UserID of creator with pseudorandom number appended to it that uniquely identifies the record.
lname	varchar(30)	Last name of individual whose information is being collected.
fname	varchar(30)	First name of individual whose information is being collected.
addr1	varchar(30)	
addr2	varchar(30)	
city	varchar(30)	
state	varchar(30)	
zip	number	
create_dt	Date	Date/time stamp created
creator_id	varchar(6)	UserID of person who created the record
modified	varchar(1)	T means record has been modified since pulled from central database
modified_dt	Date	Date/time stamp of modification
modified_id	varchar(6)	UserID of last person to edit the record

PersonData

TITLE	TYPE	DESC
person_id	varchar(30)	Foreign key indicating person described by this data.
data_field	varchar(30)	Type of data belonging to person_id. Ex cell_phone
value_field	varchar(30)	Value of the data field. Ex. 8453452314
create_dt	date	Date stored in table
creator_id	varchar(6)	UserID of person who created the record

PhotoTable

TITLE	TYPE	DESC
person_id	varchar(30)	Foreign key indicating person described by this data.
photo	blob	Photo of person matching the person_id.
create_dt	date	Date photo stored in table
creator_id	varchar(6)	UserID of person who created the record

Figure 32. Sample SQLite Data Structure

1. Sample Local SQLite Database Tables

The tables below represent data descriptions of possible SQLite database tables for the prototype created. These can be exactly the same as the external or centralized database, with minor differences in the data types of the fields. All tables in the central database need not be represented in the tables on the local devices and vice versa. That is, only required fields need to be referenced. This saves space on the mobile devices and improves processing times because developers need not allocate space and resources for unused data.

TITLE	TYPE	DESC
emp_id	text	Unique ID consisting of first and last initials of information collector with 4 digits appended to end. Must be unique to identify individual collector.
lname	text	Last name of information collector.
fname	text	First name of information collector.
create_dt	number	Date information collector was added to employee table.

Table 2. Emp Table

TITLE	TYPE	DESC
person_id	text	Unique primary key. UserID of creator with pseudorandom number appended to it that uniquely identifies the record.
lname	text	Last name of individual whose information is being collected.
fname	text	First name of individual whose information is being collected.
gender	text	One character field representing the gender of the person.
dob	number	Six digit number representing the date of birth of the individual.
create_dt	number	Twelve digit number representing the Date/time stamp when record was created.
creator_id	text	UserID of person who created the record
modified	text	Y or N indicating whether record has been modified since pulled from central database.
modified_dt	number	Twelve digit number representing the Date/time stamp of modification to existing record.
modified_id	text	UserID of last person to edit the record

Table 3. Sample Person Table

TITLE	TYPE	DESC
person_id	text	Foreign key indicating person described by this data.
data_field	text	Type of data belonging to person_id. Ex cell_phone
value_field	text	Value of the data field. Ex. 8453452314
create_dt	number	Date photo stored in table
creator_id	text	UserID of person who created the record

Table 4. Sample Person Data Table

TITLE	TYPE	DESC
person_id	text	Foreign key indicating person described by this data.
photo	blob	Photo of person matching the person_id.
create_dt	number	Date photo stored in table
creator_id	text	UserID of person who created the record

Table 5. Sample Photo Table

2. Sample Database Creation Script

```
// create emp table
// This table holds administrative data pertaining to the information collectors.
// The emp_id identifies information collector
// that either created or modified a record in the database.

CREATE TABLE EmpTable (
        emp_id          VARCHAR2(6) NOT NULL CONSTRAINT EmpTable_pk PRIMARY KEY,
        lname           VARCHAR2(30),
        fname           VARCHAR2(30),
        create_dt       DATE
);

// create person table. This table contains data collected by employees.
CREATE TABLE PersonTable (
        person_id       VARCHAR2(30) NOT NULL CONSTRAINT PersonTable_pk PRIMARY KEY
UNIQUE,
        lname           VARCHAR2(30),
        fname           VARCHAR2(30),
        dob             VARCHAR2(30),
        gender
        create_dt       DATE,
        creator_id      VARCHAR2(6),
        modified        VARCHAR2(1),
        modified_dt     DATE,
        modified_id     VARCHAR2(6)
);
```

```
// create person data table
// This tables holds fields relating to corresponding person_id in PersonTable.
// These are data fields that a person could have more than one such as phone number, vehicle, cousin, sister, etc.
// Storing this data in a separate table prevents from having to add new columns to any table in the database.
CREATE TABLE PersonData (
        person_id       VARCHAR2(30) NOT NULL CONSTRAINT PersonData_fk FOREIGN KEY,
        data_field      VARCHAR2(30),
        value)field     VARCHAR2(30),
        create_dt       DATE,
        creator_id      VARCHAR2(6)
);

// create photo table
// This table can hold multiple photos of a specific person. Using a separate table enables us to store multiple photos.
CREATE TABLE PhotoTable (
        person_id       VARCHAR2(30) NOT NULL CONSTRAINT PhotoTable_fk FOREIGN KEY,
        photo           BLOB,
        create_dt       DATE,
        creator_id      VARCHAR2(6)
);
```

3. Chapter Summary

When examining the sample SQL scripts, screen captures and algorithms described in this chapter, one must recognize that subtle changes must be made to tie all of the pieces of this project together.

The queries are for creating the actual centralized database rather than the code for creating the database on the local device. The local database on the local device was created using a datatable.java file within the android application. That code is not displayed because it varies depending on the SDKs and API being used. Personal preferences of developers play a significant role in how those tables are created as well.

As mentioned at the start of this chapter, the database architecture described in this chapter does not capture information that may be required for the devices to know the authorized users or other authorized devices with which they can communicate. Since the information required for communication varies with the communication technology or protocol being used, this focused on an algorithm that does not require a specific technology. The prototype discussed demonstrates how sharing data between mobile devices can be implemented.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

A. MAJOR ACHIEVEMENTS

The goal of this research was to identify and outline improvements that could be made to the current Lighthouse Suite of applications, specifically in how information could be shared among mobile devices in remote or challenged networking environments. To this end, research was done to identify challenges to implementing the recommended improvements and recommendations about which technologies could be levied in order to make sharing data in these environments more robust and transparent to the data collectors.

The research described the various products within the current suite of applications and how some of those products could be consolidated to provide better fidelity, improve sharing time, and possibly cut down on the overall analysis time. The most significant contribution of this research is an algorithm for database synchronization that would ensure data synchronization among devices on a local network, even when connectivity to cloud or Internet services is not available.

It involved the modification of an existing Android application created for Bluetooth Chat that allows a collector to store new information in their local database and share that information with known users within Bluetooth range. Although this prototype is fairly simplistic, it demonstrates that data collectors can collect data in these remote areas and share that data with other users of the system, without requiring actual Internet connectivity. Additionally, it walks through the algorithm and prototype to show that both function correctly and identifies areas that could be improved.

B. FUTURE WORK

1. Seamlessness

This solution used multiple applications in order to demonstrate a small portion of the capabilities required for the next generation of Lighthouse. Developing the entire application is a large task, requiring several engineers. A team of developers would be required for each piece of Lighthouse to be developed in a timely fashion. Additionally, for users to move properly from one activity to another, developers must be able to focus on specific aspects of the program. Modularity in the development of the application would allow experts in specific areas to focus on their piece of the puzzle. For example, some developers may have more expertise in map development or database management, while another may be more proficient in the visualization APIs.

2. Connectivity

Lighthouse can continue to be improved in a number of ways. This solution was developed using Bluetooth technology. More research needs to be done using Wi-Fi Direct as a means of connectivity instead. Wi-Fi Direct could mean that collectors could share information at greater distances. This capability is more powerful than the Bluetooth solution, not just because of the greater distance, but because the capacity to maintain constant connections between multiple devices is greater when using the Wi-Fi Direct technology.

Additionally, data could be broadcasted to all devices simultaneously using local IP addresses assigned to each device when connectivity is requested, rather than the pairwise sharing used with Bluetooth connectivity. This would improve the overall synchronization time as well.

3. Off-line Maps

According to their website, Leaflet is a JavaScript Library used to create interactive, mobile device “friendly” maps. It is an open source library located on GitHub developed by Vladimir Agafonkin and other “contributors” (Leaflet 2014). While GoogleMaps could be used when Internet connectivity exists, it is extremely difficult to use without connectivity. The beauty of Leaflet is the Lighthouse developers would have total control over the map tiles used. The appropriate map tiles could be loaded onto the mobile devices when loading the “mission” data. Additionally, the application could access the appropriate map tiles even without connectivity to the Internet. Figure 33 depicts a sample Leaflet map view with a blue drop pin in the center.

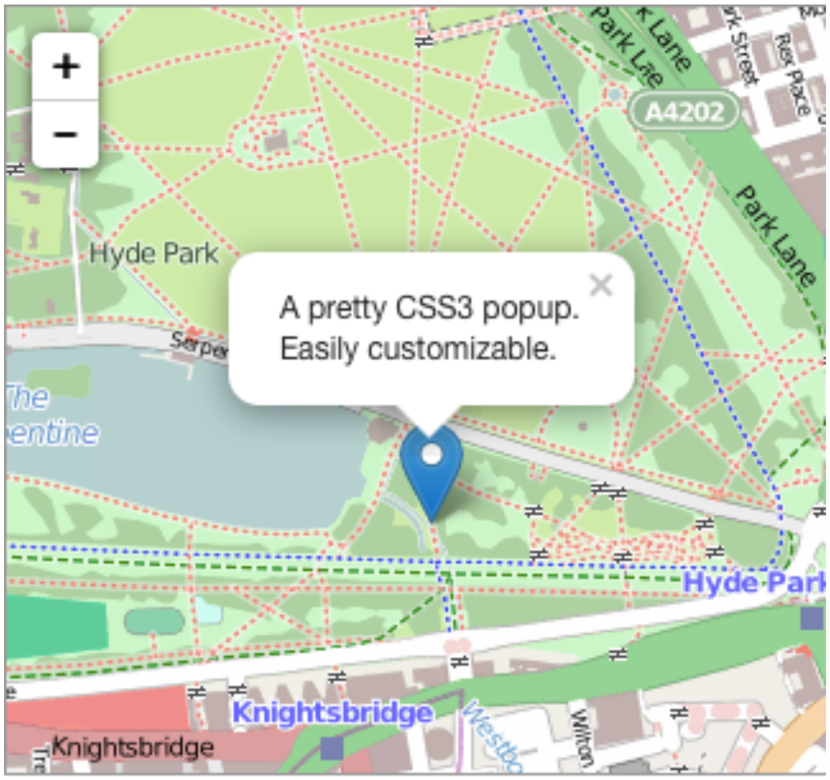


Figure 33. Sample Map View Using Leaflet (from Leaflet 2014)

4. Data Visualization

MindFusion Diagramming Library for Android is an Android Java Class Library that developers can use for providing a means for their users to visualize data. The API enables users to view graphs, flowcharts, genealogy trees and more (Mindfusion 2014). Figure 34 shows two possible views that would prove helpful to Lighthouse data collectors, depending on the goals of the collectors. The ability to create these views would enhance operations because decisions could be made regarding collection priorities at the local site rather than sending the data to a central database via the Internet and awaiting results of the analysis. This functionality would decrease the amount of time it takes for leaders at the remote site to obtain mission critical information, enhancing decision making capabilities.

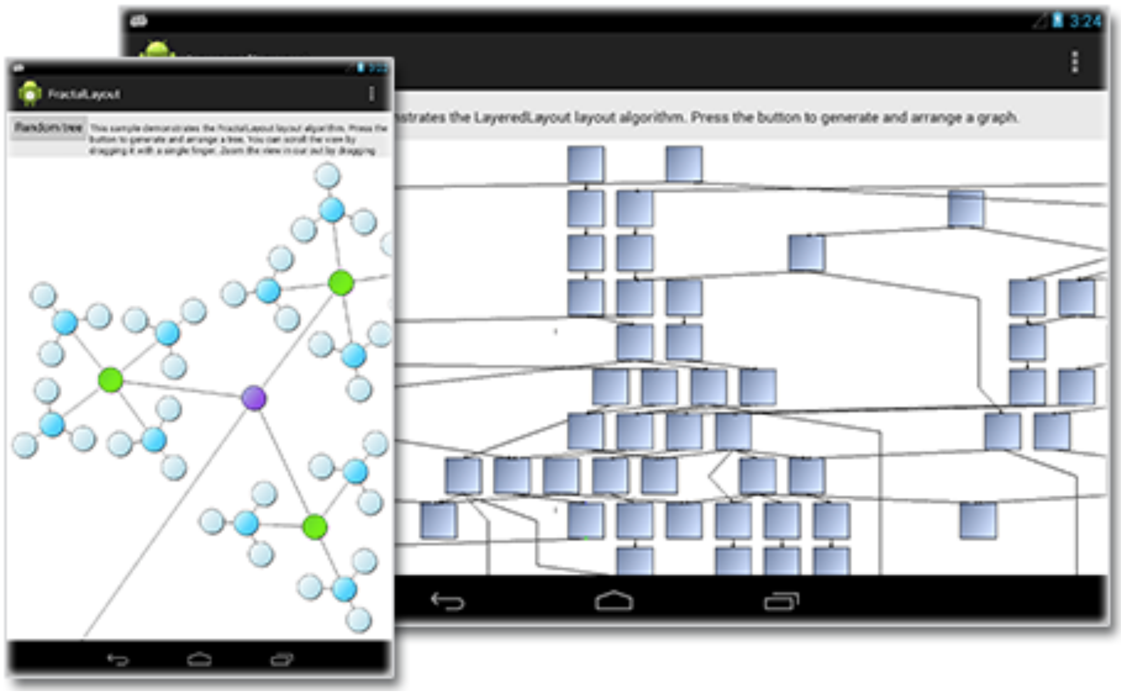

FractalLayout
Random tree
This sample demonstrates the FractalLayout layout algorithm. Press the button to generate and arrange a tree. You can scroll the view by dragging it with a single finger. Zoom the view in our out by dragging
...strates the LayeredLayout layout algorithm. Press the button to generate and arrange a graph.

Figure 34. Sample Link and Tree Diagram (from MindFusion 2014)

5. Device Independence

The original idea was to create the application using PhoneGap in order to develop a device independent application that could incorporate several tools used in Lighthouse. PhoneGap allows developers to use HTML5 and JavaScript for creating device independent applications. Basically, the same code is used for desktop or laptop web browser, iOS device, or Android device. The prototype was scaled down after multiple challenges experienced due to the learning curve and shortened timeline. Using HTML5, it proved difficult to access the local device's file system. This made working with the local SQLite database a challenge. However, if having the data stored in an actual SQLite database is not critical, the data storage could be handled using either WebDB or IndexedDB.

More research is required to test the capabilities of PhoneGap. Additional capabilities are being added to the PhoneGap libraries that enable access to specific sensors on mobile devices. If the APIs that enable access to the mobile device's Wi-Fi adapter are robust, a Wi-Fi Direct solution for connectivity could be developed. Using this, one would not have to use the local SQLite database of the mobile device. Instead, the developer would use either WebDB or IndexedDB. In either case, sharing or sending data to an external server is fairly trivial using HTML5.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Bluetooth. 2014. “Fast Facts.” Accessed June 3. http://www.bluetooth.com/Pages/Fast-Facts.aspx.

Honeggar, Barbara. 2014. “Student-Developed Smart Phone App Maps the ‘Human Terrain.’ ” Last updated June 4. http://www.nps.edu/About/News/Student-Developed-Smart-Phone-App-Maps-the-Human-Terrain.html.iTunes. 2014a. “UML Diagrams.” Accessed June 3. http://www.uml-diagrams.org/apple-itunes-uml-deployment-diagram-example.html.

iTunes. 2014b. “iTunes Store.” Accessed June 3. http://www.apple.com/itunes/features/#store.

Leaflet. 2014. “An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps.” Accessed June 3. http://leafletjs.com.

Long, Shitian. 2011. “Database Synchronization Between Devices.” Master’s thesis, KTH Royal Institute of Technology.

Longley, Carrick. 2010a. “Field Information Support Tool.” Master’s thesis, Naval Postgraduate School.

Longley, Carrick. 2010b. “Student-Developed Smart Phone App Maps the ‘Human Terrain.’ ” May 2. http://lhproject.info/student-developed-smart-phone-app-maps-the-human-terrain.

Longley, Carrick. 2011. “Becoming Familiar with the Portal.” December 18. http://lhproject.info/becoming-familiar-with-the-portal/.

Longley, Carrick. 2012a. “The Geospatial Visualization ‘Easy Button’ - Google Fusion Tables.” January 19. http://lhproject.info/the-geospatial-visualization-easy-button-google-fusion-tables/.

Longley, Carrick. 2012b. “Lab 6: Dynamic Network Analysis.” January 6. http://lhproject.info/lab-6-dynamic-network-analysis.

Malone, Aemon. 2010. “Wi-Fi Direct Mimics Bluetooth but with Faster Speed, Longer Range.” October 25. http://www.digitaltrends.com/computing/wi-fi-direct-mimics-bluetooth-but-with-faster-speed-longer-range/#!OpE7F.

MindFusion. 2014. “Diagramming for Android.” Accessed June 3. http://www.mindfusion.eu/droid-diagram.html.

MobiForms. 2014. “The MobiForms Sync Server Architecture.” Accessed June 3. http://www.aspnet.plus.com/mobiforms/mobile_sync_server.htm.

Murach, Joel. 2013. *Murach's Android Programming*. Fresno, CA: Mike Murach & Associates.

NFC Forum. 2014. "About the Technology." Accessed June 3. http://nfc-forum.org/what-is-nfc/about-the-technology.

Nori, Anil K. 2007. "Mobile and Embedded Databases." *In Proc. of ACM International Conference on Management of Data*, 1175–1177.

ODK. 2014a. "About." Accessed June 3. http://opendatakit.org/about/.

ODK. 2014b. "Aggregate." Accessed June 3. http://opendatakit.org/use/aggregate.

Oracle. 2014. " Java Message System Tutorial." Accessed June 3. http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs.

Terry, Douglas B. 2008. *Replicated Data Management for Mobile Computing*. San Rafael, CA: Morgan & Claypool.

U.S. Army Human Terrain System. 2014. "History of the Human Terrain System." Last updated April 22. http://humanterrainsystem.army.mil/history.html.

Vectorcharacters. 2014. "Vector Characters." Accessed June 3. http://vectorcharacters.ne.t

Wikipedia. 2014a. "Human Terrain System." Last modified May 26. http://en/wikipedia.org/wiki/Human_Terrain_System.

Wikipedia. 2014b. "Automated Teller Machine." Last modified May 30. http://en.wikipedia.org/wiki/Automated_teller_machine.

Wikipedia. 2014c. "Java Message Service." Last modified June 2. http://en.wikipedia.org/wiki/Java_Message_Service.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California