Cyber-physical Algorithms for Enhancing Collaboration

Daniel W. Guymon

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Christopher J. White, Chair
Thomas L. Martin
Jeffrey H. Reed

April 23, 2012
Blacksburg, Virginia

Keywords: Smartphones, Mobile Computing, Cyber-physical Systems, Security

Cyber-physical Algorithms for Enhancing Collaboration

Daniel W. Guymon

Abstract

The research presented in this thesis covers two specific problems within the larger domain of cyber-physical algorithms for enhancing collaboration between one or more people. The two specific problems are 1) determining when people are going to arrive late to a meeting and 2) creating ad-hoc secure pairing protocols for short-range communication. The domain was broken down at opposite extremes in order to derive these problems to work on: 1) collaborations that are planned long in advance and deviations from the plan need to be detected and 2) collaborations that are not planned and need to be dynamically created and secured. Empirical results show the functionality and performance of user late arrival detection for planned collaborations and end-user authentication protocols for unplanned collaborations.

Acknowledgments

First and foremost, I would like to thank my research advisor, Dr. Jules White. His guidance and continuous support has played an incredible role throughout my thesis. He helped me on all the techincal papers that have been published the past couple of years of my research. He was always patient with me, especially during my first attempts at writing, and was always enthusiastic and supportive of all my research efforts. I have learned a great deal from him about how to be a strong researcher, been able to travel around the world for conferences and have learned so much about the new and exciting field of smartphone computing.

I would like to thank the other two members of my thesis committee, Dr. Tom Martin and Dr. Jeff Reed, for their support and their time during my research. I would also like to thank Dr. Mani Golparvar-Fard for his interdisciplinary insight into my research and paper editing advice and guidance. I thank Dr. Brian Dougherty for the many hours he invested in mentoring me, guiding me through the research process, and providing me with several unique collaboration opportunities.

I also thank my colleagues in the Magnum group who have played a key role in commenting on my research and taking the time to give me valuable suggestions.

I would like to thank the Virginia Tech Department of Housing and Residence Life for their gracious support during my graduate program and whose generous funding helped make this all possible. I would like to specifically thank Donald Walker, who unconditionally believed

in me and advocated for me throughout this entire process.

I thank the National Science Foundation, who also funded by research and believed in my efforts. I am grateful for the Bradley Department of Electrical and Computer Engineering and the incredible opportunity bestowed upon me as a graduate student.

Finally, I give my thanks to my family and friends who have diligently been cheering me on the whole time. Their support was paramount in my completion.

Contents

List of Figures

Chapter 1

Introduction

1.1 Research Scope and Overview

The scope of the research presented in this thesis encompasses the problem domain of cyber-physical algorithms for enhancing collaboration. There were two types of collaboration that were investigated: 1) planned collaborations where deviations from the plan needed to be detected and 2) unplanned collaborations that needed to be quickly, dynamically and securely created. The research sub-area of power-efficient, cyber-physical late arrival detection served as a solution space for the planned collaboration domain. The other research sub-area of ad-hoc secure P2P end-user authentication protocols served as the solution space for the unplanned collaboration problem domain.

The two solution sub-areas are cyber-physical in nature; a topic I discuss futher in this chapter. Discussions on detecting lateness and automatic notifications were necessary to consider for the detection of deviations from a collaboration plan between two persons. Investigations into several different approaches to provide secure pairing protocols for unplanned collab-

orations required an understanding of end-user authentication and Bluetooth technology which is also discussed further in this chapter. Two different sets of solutions integrate these investigations and discussions in the form of cyber-physical applications on smartphones to enhance collaboration. Subsequent chapters provide in-depth formal models, test plans, empirical results and discussions for each of the solution sets.

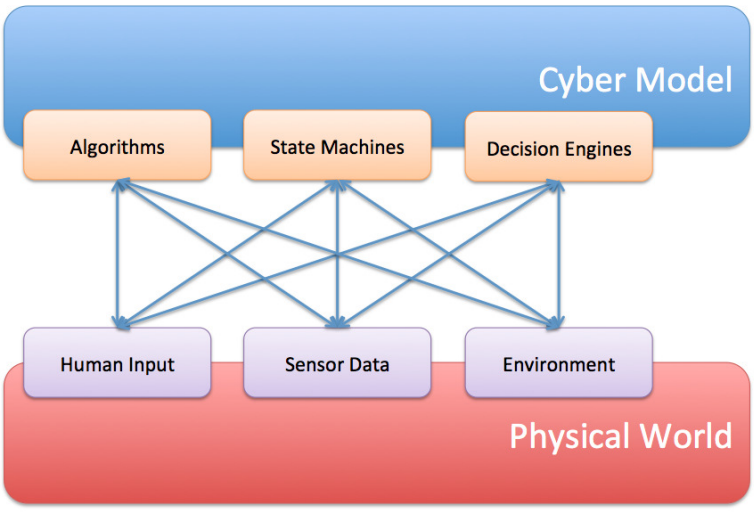
The importance of the research domain of cyber-physical algorithms for enhancing collaboration is supported and emphasized by two motivating examples in the construction industry and disaster relief areas. The solutions and data that are provided here make valuable contributions to the research domain.

This chapter is organized as follows: Section 1.2 discusses the causes and effects of lateness and a high-level definition of cyber-physical systems and Section 1.3 covers end-user authentication protocols and their use in collaborative communication. Within these sections are specifications of the problem sub-areas and solution space.

1.2 Lateness and Cyber-Physical Systems

Arriving late to an event or destination is a common and impactful problem in various domains such as mass transit, in social settings and in the workplace. Lateness can cause a delay in the start of meetings, affect employee job performance, and block further progress on a task or project. The critical impact of a persons lateness is the negative effect it has on the surrounding parties, such as co-workers and supervisors [1, 2]. For example, a person running late to a team project meeting delays the work and could potentially jeopardize the projects success if he or she is normally a valuable contributor and the deadline is near. Due to the costs of lateness, an important and relevant problem is the advanced detection of a persons lateness and the automatic notification to affected individuals.

Figure 1.1: A high-level diagram of a cyber-physical system.



Detecting a persons potential late arrival to a destination and then automatically notifying others is a cyber-physical computing problem. Cyber-physical computing is an emerging manifestation of computer-based control systems that bridge together physical occurrences and cyber calculations [3]. Specifically, cyber-physical systems change the way in which humans interact and control both the physical world and computers by using physical events generated or observed by the human as input into a cyber model [4]. Common domains of cyber-physical systems include the health care industry, power grid management and sensor networks in modern vehicles [5].

A cyber-physical system can be broken down into two components, the physical world and the cyber model. Figure 1.1 is a high-level view of the components, inputs, and outputs of a cyber-physical system.

The subcomponents of the physical world can include human input, sensor data and the state of the environment. Human input can be button presses, voice commands or physical motion and can be used as input to a decision engine or algorithm, for example. Sensors, such as accelerometers, can provide useful data for the cyber model to make a calculation about

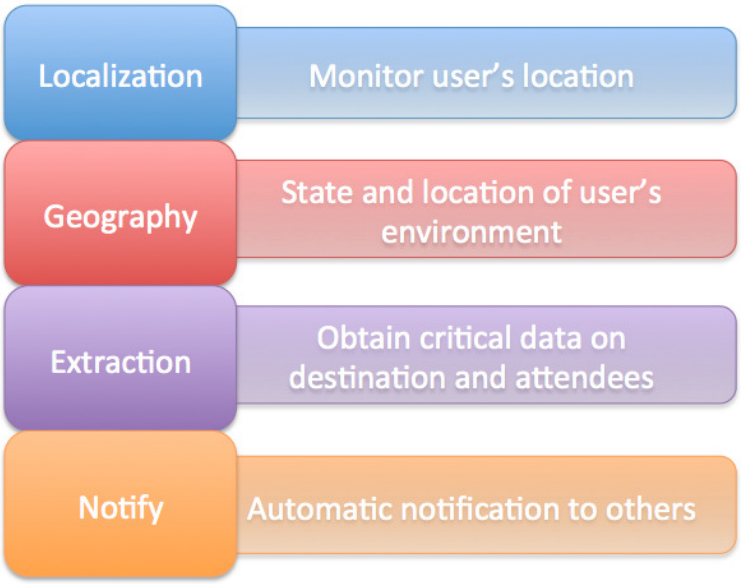
the physical world. Changes in a users physical environment can trigger a state machine to determine whether or not a person is inside or outside. The subcomponents are able to communicate to any other subcomponent, which makes the cyber model and physical world tightly coupled. An output from one component often serves as an input to another.

Pieces of information from the physical world provide the necessary input for the cyber model to calculate a users lateness, such as the users current location, current speed and the physical location of the destination. These pieces of information are made available by sensors that can monitor speed and the state of the users environment. Additionally, pieces of cyber information provided by the user, such as the start time of an event, the name of a destination, or the names of other attendees at an event, serve as input to the cyber model in determining lateness and providing automatic notification.

Given these types of physical world inputs, a cyber model can determine whether or not a user will be late with algorithms, state changes and decision trees. If the user will be late, then the cyber model can also calculate how late and then automatically notify the people already at the destination. The automatic notification can be in the form of a text message, e-mail, or some medium thatds is appropriate for the technology that the cyber-physical system is built around. Upon receiving the automatic notification, those already in attendance can then move forward with the agenda to minimize the amount of time wasted on waiting for one other person to arrive.

Figure 1.2 is a diagram of the requirements of a cyber-physical late arrival detection system. The system must first be able to monitor a users location. Information on the users current location in relation to the destination and to the users previously known location is important for the cyber model to determine whether or not the user is late. The system must also have information on the users surroundings. Data on the state of the users environment, such as indoors or outdoors, plays a key part in determining a users late arrival and user localization.

Figure 1.2: Requirements of a late arrival detection cyber-physical system.



The cyber-physical system needs to be able to extract critical information from a medium or source that allows human input. Appointment information in a calendar entry that a user manually inputs is a possible source for data extraction. Finally, the system must have features that support automatic notification to those who could be impacted by the late user.

Smartphones are a promising platform for hosting a cyber-physical system that detects and automatically relays a users lateness for several reasons. Android phones and iPhones, for example, have sensors that can monitor the users movement throughout the day. A smartphones accelerometer sensor detects acceleration in the x, y, and z-direction. The acceleration data from the accelerometer can be used by the cyber-model to determine whether or not the user is moving. The GPS on the phone traces the devices location on the earth and can provide latitude and longitude coordinates to the cyber model. The cyber model can use the coordinates to estimate the distance between the user and destination. The

cyber model can also use the coordinates to compute the users current speed. Smartphones also have a Bluetooth adapter that can be used to discover other smartphones and Bluetooth beacons in a nearby area.

The second reason why smartphones are a promising platform is because they provide a calendar interface in which the user can enter appointment information. The user can schedule the day by putting in times and locations of meetings as well as any contacts that may also be at the destination. There is also a contact list to enter co-workers cell phone numbers and e-mail addresses. The third reason is the ability for smartphones to send alert messages in the form of text messages and e-mails. Using contact information extracted from the contact list, the smartphone can automatically set up a message informing the contact of the user's late arrival.

1.2.1 Problem Domain: Power-efficient Smartphone-based Late Arrival Detection and Automatic Notification

Even though there is great potential for using smartphones as a cyber-physical platform for late arrival detection, there is little prior research approaches for detecting lateness and automatically notifying others using smartphones. User localization is one of the major components of detecting late arrivals, and a number of researchers have investigated various approaches to user localization [6–12]. None of these approaches have addressed all the challenges related to late arrival detection and automatic notification on smartphones, though.

Smartphone-based methods of automatic notification are the second component of user lateness detection. Several researchers have investigated different methods of automatic notification in response to an event on smartphones [13]. These solutions, though, have not addressed all the challenges related to automatic notification of a users late arrival on smart-

phones.

There are four key cyber-physical issues that have not been addressed for late arrival detection on smartphones. The first issue is that localization sensors on smartphones consume significant power and can exhaust a smartphones battery in only a few hours. For example, using GPS on a smartphone drastically shortens how long the device can be continually used before it requires to be recharged. The second issue is that many indoor localization approaches rely on external hardware, pre-analysis of wireless signals in a building, or other environment conditioning steps to determine physical information. Steps to setup an environment for indoor localization with beacons, floor plans and continual human input may not be economical or possible to perform at all locations a user travels to.

The third issue of late arrival detection is the requirement for both outdoor and indoor localization, which requires an approach that can seamlessly transition between the different localization approaches used in each environment. As a user moves from one location to another for a meeting, the user's environment can change from indoors to outdoors and back again. The fourth cyber-physical issue is that once the user enters the building, the GPS location lock is lost so calculating the estimated time of arrival becomes more difficult, especially if the destination is in a rather large building that requires some time to navigate.

1.2.2 Solution Overview: A Cyber-physical Approach for Detecting User Lateness

To fill the gap in research on smartphone-based cyber-physical late arrival detection, I have developed a smartphone-based late arrival detection and notification approach called PENDING, PrEdictioN of Delay in arrIving at a meetiNG.

PENDING uses smartphone sensors to determine a user's physical context through localization and environment assessment, more intelligently power-cycles localization sensors,

significantly reduces the power required to detect a late arrival and provides automatic lateness notifications. Moreover, PENDING uses a key insight from the domain of late arrival detection, that indoor localization only needs to determine if the user is in a meeting room. The insight simplifies the indoor localization component of late arrival detection.

One key attribute of the PENDING approach is that it requires significantly less power to detect late arrivals versus other approaches because the intellegent use of changes in the user's environment and how long user localization takes place. This attribute reduces the time that localization sensors are unnecessarily active. A second key attribute of the PENDING approach is the high fidelity in detecting late arrivals. In our results section, we present empirical data that we gathered from real-world experiments that tested the operation of PENDING and compared its power consumption to late arrival detection with existing localization approaches. The empirical results show that PENDING uses up to 42% less power than late arrival detection with existing localization approaches. In addition, our results show that PENDING is capable of seamless indoor/outdoor localization handoff and a 90% success rate in detecting late arrivals.

1.2.3 Contributions to Late Arrival Detection

My work on PENDING makes the following contributions to the study of late arrival detection with smartphones:

- It presents a novel localization technique for mobile devices that is capable of determining device location seamlessly between indoor/outdoor environments without requiring additional preconfigured hardware.
- It provides a formal methodology for the acquisition of physical data from smartphone sensors, such as device location and nearby device IDs and relating it to a cyber

model that contains schedule data, such as appointment times, locations and contact information, to detect late arrivals.

- It presents empirical results showing that PENDING

  significantly reduces power consumption versus other purely-GPS and purely-Bluetooth mobile device localization approaches.

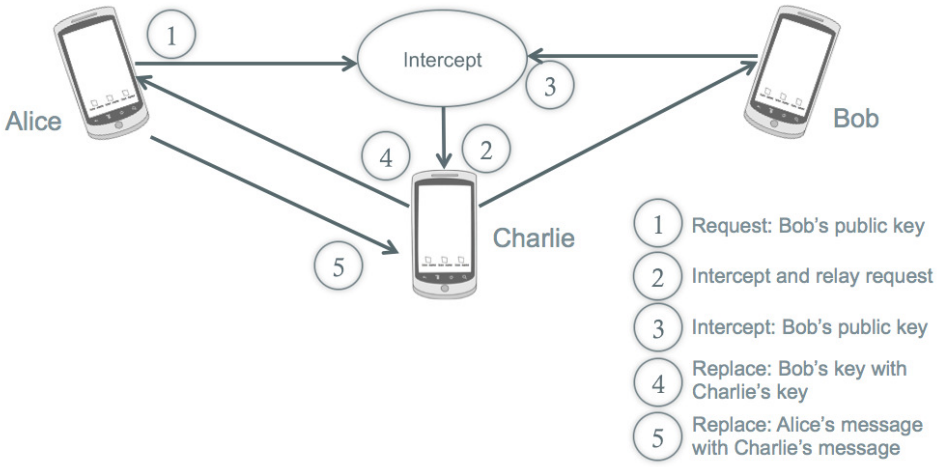
- It shows that PENDING provides a robust notification algorithm that adapts to the user's current surroundings.

1.3 End-User Authentication

Cyber security is a growing concern in collaborative communication, such as crowdsourcing and P2P file transfers, because of malicious users' ability to exploit security vulnerabilities of the communication channels. Attackers can disclose, change and block access to sensitive information by other users [14]. Peer-to-peer Bluetooth-based communication also suffers from the same types of attacks, even on smartphone platforms [15, 16].

One specific vulnerability is end-user spoofing, or masquerading. Spoofing occurs when a malicious third-party is able to extract improperly handled pairing information, such as keys and device IDs, and use it to create a trust relationship with one or both of the other users [17,18]. A malicious user exploits this vulnerability by employing a man-in-the-middle attack as seen in Figure 1.3. One user, Alice, initiates a public key request from another user, Bob. Charlie, who is able to intecerpt messages from both parties, intercepts and relays the request to Bob. Bob sends his public key to Alice in a response message, which Charlie intercepts and acquires Bob's public key. Charlie alter's the response and provides his public key to Alice. Alice generates a message and encrypts it with Charlie's public key rather than

Figure 1.3: Diagram of a man-in-the-middle attack.



Bob's. Charlie intercepts the enciphered message, decrypts it, alters it and then sends it to Bob. Charlie, who is now the man-in-the-middle, is able to intercept and alter any message that is sent between Alice and Bob.

Other attacks include Bluesnarf attacks that can access restricted areas of a user's Bluetooth-enabled device through vulnerabilities in the Bluetooth Object Exchange protocol [19]. Another attack, called the Backdoor attack, exploits the Bluetooth pairing protocol by removing the paired connection with a malicious device from the paired devices list on the user's device. The small, but significant, omission makes it difficult for users to detect if their devices is accidentally paired with a malicious device unless suspicious activity is noticed while it is occuring. Because of the widespread use of Bluetooth-based collaborative communication, determiningg efficient mechanisms for authentication and security on these channels is critical.

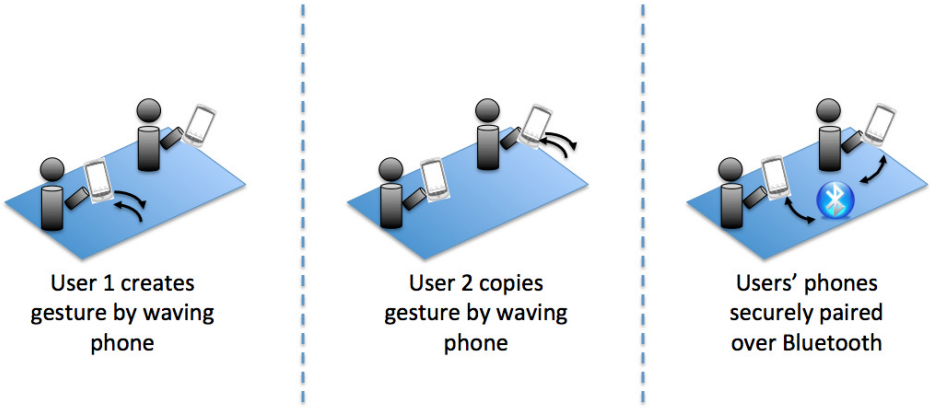
End-user authenticaiton when exchanging information between devices using Bluetooth is an important concern. The Android Bluetooth health device profile is a new API for Android smartphones to be able to securely monitor health data and transmit sensitive medical

information to each other, as defined by the security requirements set forth by Bluetooth SIG [20,21]. In order for any data to be transferred, an end-user's unique identity must first be established and then authenticated by the other end-user to open a trust relationship over a peer-to-peer (P2P) communication channel. Without authenticating the end-user's identity, sensitive information can be stolen. For example, Bluetooth packet snooping programs, like hcidump, monitor communication channels and are able to steal packets of sensitive information if the channels are not protected [22]. Bluebugging is another dangerous attack on Bluetooth-enabled mobile phones where the attacker gains read/write access to the phone's contact list, e-mail and text messages [23].

Establishing the identity of the user that is at the other end of the Bluetooth connection requires some form of preconfigured identity or out-of-band communication of authentication information. For example, users can be preassigned user names and passwords that can be used to verify identities. However, when there is no preestablished trust relationship between users, such as in a disaster response scenario where a citizen needs to share data with a first responder, out-of-band communication mechanisms are used to exchange secret authentication information. For example, one user may generate a unique Bluetooth PIN and verbally communicate it to the other user in order for the second user to establish his or her device's identity.

Due to the limited typing speed of users on touchscreen on mobile devices [24], the out-of-band secrets used for authentication are often short (e.g. 3-4 numeric digits) to make input easy on a mobile device. However, shorter authentication secrets can lead to less secure communication channels. Because typing even 3-4 numeric digits can be cumbersome on a mobile device [25], many users forgo PINs or simply keep the factory default number [26] and blindly establish connections to other devices that they believe a friendly end-user.

Figure 1.4: Diagram of end-user authentication with gestures.



1.3.1 Problem Domain: Ad-hoc Secure End-User Authentication for P2P Bluetooth Communication on Smartphones

An alternative approach to PINs for Bluetooth identity establishment is to use the movements of the device as a secure token that can be replicated by the intended target user. For example, one user can wave the device in the air using a unique motion that can be duplicated by a second user to establish his or her identity. Smarphones, such as the iPhone and Android phone, are a promising platform for providing this type of motion-based end-user authentication through their built-in accelerometers and gyroscopes. The phone's accelerometer can be used to measure motion gestures to generate a shared unique key with equivalent or greater security (e.g. more bits) and easier input (e.g. more bits entered per second) than traditional pin-based Bluetooth authentication approaches.

Despite the great potential for using smartphones for secure ubiquitous communication, little research has been done on approaches for end-user authentication protocols using the smartphone's accelerometer. A number of researchers have investigated approaches for gesture recongition on smartphones, which is a major component of gesture-based authentica-

tion [27–29]. However, none of these approaches has addressed all of the challenges related to gesture-based end-user authentication on smartphones.

There are three key challenges that need to be addressed for gesture-based end-user authentication on smartphones. The first challenge is the trade-off between security and usability of an end-user authentication protocol, which can make it difficult to establish a trust relationship between two users. The more precise the motion is tracked, the more bits can be generated for the authentication secret. However, more precise tracking of the user's motions also makes it more difficult for the second user to exactly replicate the original gesture. The second challenge is that the raw data of the smartphone's accelerometer is jittery and can make generating a key that can be replicated through motion on a second device hard. The third challenge of gesture-based end-user authentication is the difficulty in discriminating between trusted and untrusted users that may also be able to see the first user performing the original gesture. These challenges are covered in detail in the next chapter.

1.3.2 Solution Overview: Gesture-based End-User Authentication on Smartphones

To fill this gap in research on smartphone gesture-based end-user authentication protocols, I investigated, implemented and tested several approaches to protocols for P2P Bluetooth smartphone end-user authentication using accelerometer-based gestures. I explored the usability, security and speed tradeoffs of each protocol by running speed tests, gesture-detection tests and calculated the relative security of each protocol. Four protocols were considered and are each based upon a different algorithm: (1) jerk differentiation; (2) software debouncing; (3) acceleration-matching with sliding window, (4) and gesture template thresholding. The algorithmic details of each of these protocols will be discussed later.

Each of these protocols use the smartphone's accelerometer to monitor unique handshake

gestures the users make to establish an identity that can be shared and verified with each other. The protocols use a key insight from the domain of gesture-based P2P end-user authentication on smartphones that smartphones have computing power and are able to process motion gesture data to allow for gesture comparison between two smartphone users.

1.3.3 Contributions to End-User Authentication

My research makes the following contributions to the study of gesture-based end-user authentication for P2P Bluetooth communication on smartphones:

- It presents four novel pairing techniques for smartphone collaboration using accelerometer gestures that provides varying levels of security against end-user masquerade attacks.
- It presents empirical results showing the operation and security of each of the four protocols which provide equivalent or greater security than Bluetooth pin numbers and equivalent or faster input.

1.4 Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 provides dicussions on the motivation and challenges of the research. Chapter 3 provides a literature review of some related work to the research topic. Chapter 4 provides on overview of our solution to late arrival detection, PENDING. Chapter 5 discusses the test plan and empirical results of PENDING. Chapter 6 covers each of the four protocols for end-user authentication using handshake gestures. Chapter 7 describes the test plan and empirical results of each of these

protocols. Chapter 8 discusses the results in further detail and Chapter 9 provides concluding remarks.

1.5 Related Publications

Several related publications were reviewed and published as a basis for this thesis:

1. Power-efficient Cyber-physical Late Arrival Detection with Smartphones in IEEE Conference Proceedings on Cyber, Physical, and Social Computing, CPSCom 2011, Daniel Guymon, Jules White, Brian Dougherty, Mani Golparvar-Fard.

2. Power-efficient Cyber-physical Late Arrival Detection with Smartphones in 2012 International Special Issue Journal on Context-Aware Mobile Systems, Daniel Guymon, Jules White, Brian Dougherty, Mani Golparvar-Fard (to appear).

3. Accelerometer-based Gesture End-User Authentication Protocols for P2P Bluetooth Communication on Smartphones, 4th International Conference on Future Computer and Communication, ICFCC 2012, Daniel Guymon, Alan Baines, Jordan Schweigert, Jules White. (submitted)

Chapter 2

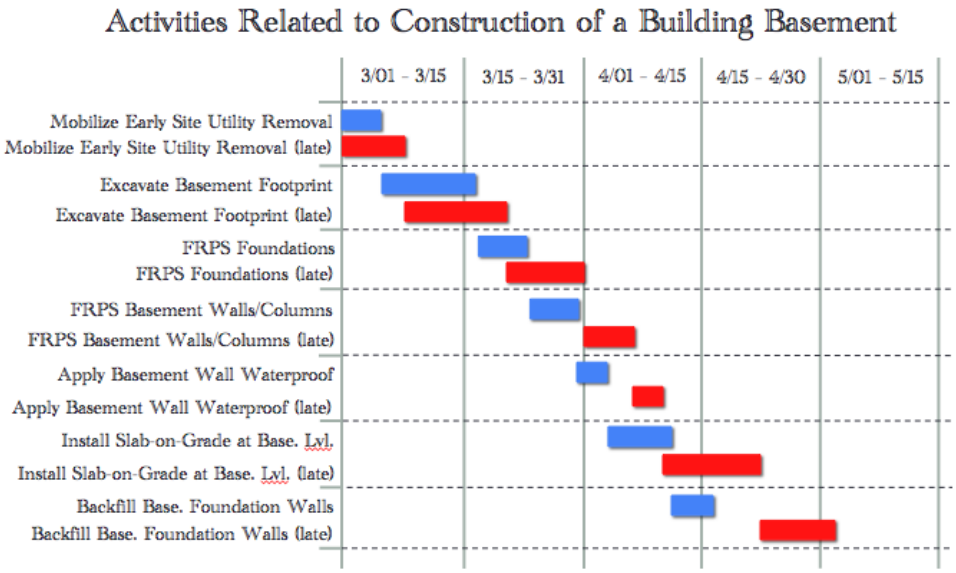
Motivation and Challenges

2.1 Motivation for Late Arrival Detection on Smartphones

As a motivating example to illustrate the challenges of late arrival detection, we use a scenario from the Architecture, Engineering and Construction (AEC) industry. Late arrivals of construction resources (i.e., labor or material) are a significant source of delay on construction projects, which can cause significant delays in the overall schedule of a project, and negatively impact the total cost [30, 31].

A critical issue on construction sites is that many construction activities are directly interlinked and dependent on the completion of preceding tasks. For example, as seen in Figure 2.1, initial forming of concrete basement foundation is dependent upon the completion of the foundation. Despite the importance of timely arrival of construction workers or delivery of construction materials, there are often cases where either a particular trade of construction workers (e.g., plumbers) are late, or the material in need for completion of a

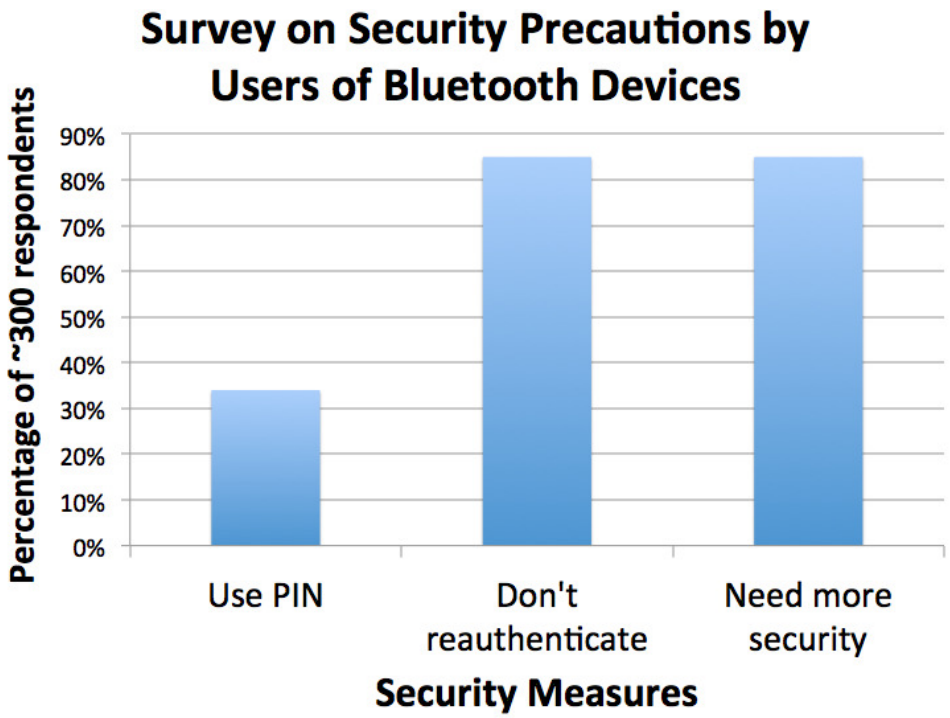
Figure 2.1: An example schedule of dependent construction tasks.



task is not delivered on time.

If it was possible to predict when a contractor was going to be late, other workers on site could be assigned to alternative tasks that are not directly dependent upon the late contractor or delivery of the material. Studies have hown that in a sample of 215 commercial and industrial construction projects, 62% of the projects were behind schedule and 88% of these late projects were behind due to the late arrival of contractors and their associated construction crews. In most of these projects, the overall project schedule was extended by atleast a month, and in several extreme cases, the project was six months behind schedule [32]. Such delays can have monumental costs to the customer funding the construction project.

Figure 2.2: Bluetooth security survey responses.



2.2 Motivation for Gesture-based Authentication on Smartphones

A key problem with existing Bluetooth authentication mechanisms is that users often fail to employ them effectively. In a survey of roughly 300 respondents who regularly use Bluetooth technology, 34% did not use a unique PIN to secure their Bluetooth-enabled devices [33]. Of those that did use a PIN, 85% left the devices on all day without ever reauthenticating, thus leaving an open channel and increasing the possibility for malicious users to attack. Despite the relatively high misuse and lack of understanding of Bluetooth security, 85% of all respondents saw a need for increased end-user authentication security beyond PINs and passwords as seen in Figure 2.2.

Security is an important concern in disaster relief or healthcare scenarios where emergency

personnel often share sensitive information on victims, such as medical records [34]. The largest security concern of smartphone collaboration in disaster areas is untrusted users masquerading as trusted users. Without a secure pairing protocol, an untrusted user could gain access to the communication channel and acquire data not intended for public eyes. At the same time, speed of authentication is critical in these domains requiring that any authenticaton mechanism be easy for personnel to employ.

If it were possible to authenticate a user's identity with handshake gestures to meet both the security needs of the medical personel as well as the usability needs of the 34% of users who do not use a unique PIN to secure, then more collaborative smartphone communication could be done.

2.3 Challenges of User Lateness Detection

Although smartphones are a promising platform for user late arrival detection, there are several challenges that must be addressed related to the limited battery capacity of smartphones, automatic notification, estimated time of arrival and issues associated with indoor localization. Because phones are usually only charged at night when the user is asleep and many meetings occur indoors, power efficiency is critical for developing a smartphone late arrival detection system. Moore's Law defines the rate of advancement for certain computing attributes, such as processor speed and storage density. However, battery power density has not increased nearly as fast as processor speed [35].

2.3.1 Power Efficiency Challenge

A key challenge of detecting late arrivals using smartphones is that smartphones have limited battery capacities that can quickly be exhausted using smartphone localization sensors, such

Figure 2.3: The effects of GPS on smartphone battery life.

Smartphone	Battery Life (mWh)	GPS (140 mW)
iPhone 4	1420	10.14 hrs
Droid X	1570	11.21 hrs
Nexus One	1400	10.00 hrs
Galaxy S	1500	10.71 hrs

as GPS. For example, the iPhone 4, the Droid X, and the Nexus One have 1420 mWh, 1570 mWh and 1400 mWh battery capacities, respectively. A GPS sensor, alone, consumes roughly 140 mW of power when enabled [36]. When neglecting all other power consumption on the phone, the battery is expected to last 10 - 11 hours for the iPhone, Droid X and Nexus One when GPS is turned on, as seen in Figure 2.3. In reality, an active GPS sensor also requires the CPU and other power consumption resources. When these other supporting resources are considered, battery life may be as short as 3-4 hours when GPS is enabled. Because users normally charge their phones at night while they sleep, running GPS all of the time is not a feasible approach for localization.

In the context of our construction scenario, described in Section 2.1, the power consumption challenge becomes an issue when a late arrival detection application on the worker's smartphone uses up too much battery power throughout the day. The average construction worker's shift is eight hours, so in order to benefit the construction domain, an algorithm must not exhaust a smartphone's battery in less than this amount of time. Once the battery is depleted, late arrival detection of the smartphone's owner comes to a halt until the battery is recharged. Since most people recharge their phones at night while sleeping, it is not likely that they would be able to recharge their phone during the day at work on a construction site. This challenge is addressed by using a smartphone's accelerometer to determine user context and intelligently enable and disable GPS and other localization sensors to save power.

2.3.2 Challenge of Indoor Localization Without External Hardware

The second challenge of smartphone late arrival detection is indoor localization without external hardware, such as Bluetooth beacons, WiFi hotspots and GPS repeaters. Indoor localization is the process of monitoring a person's location in a building. For late arrival detection, indoor localization must be accurate enough to determine if a user is in a specific meeting room or not. Outdoor GPS localization in an indoor environment is inadequate due to physical barriers blocking signals. Outdoor GPS in an urban environment is another practical issue, but is not within the scope of this project.

Many previous indoor localization approaches have relied upon preconfigured external hardware, such as custom-made Bluetooth beacons or wireless routers and GPS repreates, in order to track a user's location indoors. Each of these approaches requires reconfiguration of the indoor environment, careful calibrations or additional hardware. For example, in WiFi localization approaches, the cell signals through the building must be mapped and the area of overlap between access points measured. In Bluetooth localization approaches, Bluetooth beacons are laid about a floor plan requiring custom-made containers to focus the signal for increased detection range. In accelerometer-based indoor localization, floor plans are downloaded to the mobile device and the user specifies a starting location. Extra hardware increases the cost and preconfiguration of the environment and may not be practical due to the lack of robustness and portability to other environments.

Referring back to our motivating example in Section 2.1, construction workers already have to use and track a large number of construction equipment and tools, so the introduction of fragile and expensive headsets of beacons for late arrival detection and localization would not be ideal in harsh construction environments. Also, due to the nature of construction operations, sites are dynamically challenge and as a result, approaches that rely on measure-

ments of external physical phenomena, such as WiFi signals, would have to be continually recalibrated as the building is being built and the signal propagation properties on the site is changed. I address this challenge by using the smartphone's Bluetooth adapter for indoor localization without reconfigured external hardware whose details are discussed later.

2.3.3 Seamless Indoor/Outdoor Localization Handoff Challenge

The third challenge of user late arrival detection is the difficulty in accomplishing seamless indoor/outdoor localization transitions. Seamless transitions are necessary because they ensure continual location tracking regardless of the changes in the surrouding environment. Unfortunately, there is no single sensor that specifically accomplishes seamless localization transitions and outdoor GPS is not adequate for indoor GPS use [10, 11]. While there have been many research efforts focused on indoor and outdoor localiztion exclusively, there are few that seek to accomplish seamless handoff between the two localization methods. One solution, called Streamspin, uses a combination of GPS and WiFi hotspots to acheive seamless indoor/outdoor localization handoff on a mobile PDA device [37].

Even though Streamspin accomplishes seamless indoor/outdoor localization transitions, the algorithm depends upon GPS, requires preconfigured WiFi maps and consumes 35% of the device's battery per hour, which exhausts the battery too quickly for many late arrival detection domains. Note that this challenge is not mutually exclusive from the previous two, thus making seamless indoor/outdoor localization that much more difficult to address. One may be able to accomplish seamless localization handoff, but the accomplishment often comes at the expense of power consumption and simplicity of design.

Construction workers may walk in and out of the building during construction. In order to be able to track the late arrivals of workers or of specific job-related tasks, it is necessary to accomplish seamless indoor/outdoor localization handoff in a user late arrival detection

algorithm. For example, a plumber is going to work on several construction projects in the same day. On some sites, he may be doing plumbing work inside a partially finished structure where the GPS fix on his location may be periodically lost and found. At other parts of the day, he may be installing pipe as part of the foundation for a building and has a continual GPS location fix on him. A late arrival detection algorithm with seamless localization would be able to keep track of where the plumber was at any point of the day in order to determine if he was running late to a job based upon his work schedule. I addressed this challenge by developing a GPS-based algorithm for detecting indoor/outdoor localization handoff whose details are discussed later.

2.3.4 Challenge of Estimating Arrival Time and Social Issues of Lateness

The fourth challenge concerns the issues of providing accurate estimated arrival times and social issues of predicting user lateness. Upon the user entering a building, outdoor localization is no longer effective for maintaining a location fix while the person walks around the building. Estimating arrival time requires location udpates in order to calculate the speed at which the user is walking and how far away he or she is from the destination. Presently there are no prior solutions that address the specific challenge because of the non-constant variables for each building and user such as the layout of the destination and the indoor walking speed and stair-climbing speed of the user.

Accurately predicting a user's lateness is generally a difficult task. Some social issues, such as emergency phone calls and unexpected visits from supervisors, can affect how late a person will be to an appointment. Common causes of lateness, though, stem from attitudinal and professional immaturity, lack of urgency, lack of respect for other people's time and time spent commuting to work [38,39]. There are also privacy concerns of providing user location

data to other people, but we did not focus on the specific issue in our research.

Relating back to our motivating example for user late arrival detection, as a contractor moves from site to site and from indoors to outdoors, his location fix will inevitable be unavailable at times throughout the day. The shifts in environment may cause inaccuracies in estimating arrival times throughout the day to different jobs. A user lateness detection algorithm must be able to accomodate for these eventual changes and still provie some level of arrival time estimation. For example, if a contractor has severl different tasks to complete at several different job sites, a user lateness detection algorithm could provide status updates while en route, but once he enters the building, estimating how late he is to the next job becomes difficult. This challenges is addressed by using the automatic notification feature the moment the user moves from outdoors to inside the destiantion building.

2.4 Challenges of End-User Authentication on Smartphones

Although motion-based authentication provides a promising solution to the scenarios outlined in Section 2.2, there are a number of challenges to developing an end-user authentication protocol using handshake gestures on smartphones. The first challenge is achieving a balance between usability and security. An end-user authentication scheme that is too stringent makes ubiquitous smartphone collaboration difficult because the second user will have a harder time copying the gesture created by the first user, while a scheme that is too lenient makes collaboration too insecure. The second challenge is that data from smartphone accelerometers is jittery without any processing or smoothing, which makes the direct comparison of two sets of raw data somewhat meaningless. The third challenge is the difficulty in deriving appropriate gesture matching thresholds, which are necessary to compare the similarity between two users' gestures.

2.4.1 Challenge of Balancing Usability and Security

The balance between usability and security of a communication protocol depends upon several factors. Liu et al. state that a gesture-based end-user authentication scheme should be easy enough for the average user to understand, but also secure enough to communicate sensitive information [27]. A gesture-based end-user authentication scheme whose gesture matching requirements are too strict makes it difficult for a user to properly establish their identity because the variability of motions can be very large.

Relating back to our disaster relief example in Section 2.2, when two medical personnel need to share a medical record of a victim, they will want to be able to do so quickly and securely with motion gestures. Implementing a gesture-based authentication algorithm that is too strict or complex will hinder their ability to do their job and tend to other victims, while a too lenient algorithm opens the door for unauthorized access to the communication channel. Ideally, a quick handshake gesture should be enough to generate a secure identity that each user can authenticate with. Section 6.1 describes how we address this challenge by catering our end-user authentication algorithms to use accelerometer data generated by physical motion gestures on the smartphone.

2.4.2 Jittery Accelerometer Data Challenge

Even though the hardware features on smartphones are provide useful data, there is an known issue with the data collection accuracy of the accelerometer, for example. Other related research efforts utlizing the same technology show that on-board smartphone sensors are susceptible to noise causing an observed jitter in data readings even when the device is stationary [40]. Preprocessing raw accelerometer data is necessary for the readings to have any mathematical value [41]. The challenge lies in the amount of preprocessing to be done

on-board the smartphone.

If the disaster relief personnel described in Section 2.2 use physical gesture motions on their smartphones to authenticate their identities, then the device's accelerometer would be the primary source of data for gesture comparison. If the raw data from the two phones were to be directly compared, then it would be nearly impossible to establish a match even if the users perform seemignly identical gestures because of the jitter. Some level of preprocessing must take place on the phone in order to smooth out the data for more appropriate gesture matching, otherwise the personnel would be delayed in trying to generate a gesture match. Section 6.1 describes how we address this challenge by applying data smoothing algorithms to the raw data provided by the smartphone's accelerometer sensor.

2.4.3 Gesture Matching Threshold Parameter Derivation Challenge

Discriminating between a malicious user and a trusted user solely on a physical gesture can be tricky because of the wide variability in possible movements. The challenge lies in determining particular threshold values that can serve as cutoff points for comparing gestures. By setting the threshold too high, the library of acceptable gestures increases as does the possibility of a gesture counterfeit. By setting the treshold too low, the usability becomes difficult.

In regards to our motivating example, deciphering between trusted users and unauthorized users is paramount in ensuring data security of medical records in a disaster relief scenario. There is a possibility that a malicious user could be aware of the gesture-based authentication scheme, so the algorithm must be unique enough so that the untrusted user would be unsuccessful in trying to mimic a particular gesture.

Chapter 3

Literature Review

3.1 Late Arrival Detection Related Work

Accurate methods of indoor and outdoor localization and estimating arrival time comprise a significant portion of the issues in detecting late arrivals. I provide here a taxonomy of related work that I divide into eight categories: (1) indoor WiFi localization; (2) indoor accelerometer with floor plan localization; (3) indoor Bluetooth localization; (4) probabilistic indoor localization; (5) outdoor GPS localization; (6) seamless localization handoff with technology fusion; (7) GPS power-cycling; and (8) estimated time of arrival.

3.1.1 Indoor WiFi Localization

Indoor WiFi Localization is the use of WiFi hotspots and WLAN mapping to acheive indoor location tracking down to a relatively small distance, such as between offices in a building. One of the solutions that uses indoor WiFi localization is Streamspin, which is a mobile localization application used on PDAs [37]. The application compares the signal strengths

of the wireless routers in the building to a precompiled database of signal strength-distance pairs stored for each wireless point. As the user walks around, the application on the PDA takes readings of the emitting WiFi signal and does a comparison with the data in the database. If the device can match the signal strengths with the multiple routers in the database, then the device is able to triangulate its indoor location. Streamspin offers a publish/subscribe architecture where vendors, such as restaurants and stores, can push content through WiFi routers to client devices whose users can be seen walking around the premises. Devices could then ubiquitously download coupons or daily lunch deals of the location.

Lifetag is another WiFi-based localization solution that is very similar to Streamspin [9]. Lifetag is a "life-logging" solution where a person's activities are continually logged based upon user localization. Instead of a PDA, the user carries a WiFi sensing device that only collects access point IDs and signal strength at the moment of collection and then compares these pairs to those in a precompiled database to map where the user is at in an urban environment or academic building, for example. The database the researchers used included roughly half a million access points and their associated locations within Tokyo, Japan. With the logs, Rekimoto et al. were able to construct cognitive and density maps of locations within Tokyo that had the most visits. The logs can also be used as an activity prediction engine for a single user during a week. Any localization that is outside the norm of a person's travel can give valuable information as to whether or not the person should be at a particular location.

Directional beaconing is another WiFi based indoor localization approach that utilizes custom made rotation WiFi access points whose direction of signal emission is constantly changing [42]. Kawauchil et al. constructed rotating WiFi beacons whose emission direction would change with the rotation.

Rather then using angle-of-arrival or time-of-arrival calculations, they used angle-of-emission calculations to localize a device based upon whether or not the device could detect the signal and the angle at which the beacon was facing. Using this methodology eliminates the need for extra sensing hardware beyond the mobile device and WiFi beacon.

3.1.2 Indoor Bluetooth Localization

Indoor localization can also be acheived with Bluetooth. Cheung et al. accomplish Bluetooth location tracking using beacons and a smartphone. [8]. Custom-made short-range Bluetooth beacons were constructed and all set to discovery mode so that the smartphone navigating through the maze of beacons could keep track of its current location. The low-cost, of-the-shelf solution is ideal for indoor localization whether room entry detection is sufficient, such as in a workplace.

Many previous indoor Bluetooth localization solutions depend upon RSSI value mapping, but as Bargh and Groote discovered, the difference in Bluetooth technology in different devices creates a non-uniform correlation between RSSI values and distances [43]. Because of this observation, they focused on indoor bluetooth localization at a room level using an approach called Context Management Frame. Their approach uses inquiry response rates of Bluetooth sensors to localize stationary mobile devices within an office building environment. The accuracy of their system is 98% when the device is stationary for three minutes.

Fischer et al. accomplish indoor Bluetooth localization by measuring the difference in time-of-arrival of signals between a mobile device and four stationary Bluetooth stations [44]. They also use a correlation integrated circuit and microcontroller to do the calculations. The basic principle is that when a mobile device sends out a Bluetooth discovery signal, the four stationary Bluetooth receiver stations will detect that signal at different times since they are different relative distances away from the mobile device. Using the differences in time

and knowing the relative distances the receivers are away from each other, then 1m-level localization accuracy can be obtained.

3.1.3 Indoor Accelerometer Localization

Some researchers have even explored the possibility of using an accelerometer as a means of indoor localization. Hsu and Yu transform data from an accelerometer to velocity and direction vectors to accomplish coarse-level indoor localization given a starting location within the building [45]. Their system used simple, low-power calculations to translate speed of a remote control car equiped with an accelerometer to displacement vectors in order to localize the car in a room. Their findings showed that the higher the frequency of accelereometer data readings, the more accurate the car could be localized.

Some methods of indoor accelerometer localization use a dynamically constructed floor plan of the indoor area being navigated to cross reference to track a user's location within a building [46]. SmartSLAM is an indoor localization solution that uses the smartphone's accelerometer, gyroscope and awareness of nearby WiFi access points to digitally construct a floor plan of the indoor area. The approach uses statistical analysis to process the accelerometer data into velocity vectors and the gyroscope data into directional vectors to build the floor plan and synchronize with the WiFi mapping monitored by the phone. While the device moves around the building, the combination of accelerometer data and gyroscope data transformations constructs a path that can be digitally seen on the smartphone screen. WiFi monitoring provides any necessary position recalibration if the path veers too far off track.

Ofstad et al. observed that using GPS to localize a user to an indoor area can result in hugh errors, such that Google Maps could determine that a person is in one location when the person is actually a city block away [47]. By using a smartphone's accelerometer to

augment indoor localization context, eggregious positioning errors can be mitigated. The researchers were specifically interested in whether or not the smartphone's user was standing or sitting, which gave more accurate context as to where the user was. In a controlled study using locations where the user would either be sitting or standing, the proposed system was able to correct GPS-localization errors 98% of the time by using the accelerometer data to establish context.

3.1.4 Indoor Probabilistic Localization

Some researchers employ statistics and probability models to predict where a target is in an indoor environment. Thrun et al. applied their understanding of probability models to localize autonomous robots in indoor settings [48]. By building a history of significant events, directional bearings and travel speed, one can use probability models to predict the location of a robot in a given location. The researchers constructed a history by manually driving a robot around an area and recording any physical obstacles that were in the way while traveling. The history could then be used by the robot to autonomously navigate through the same area with minimal human intervention.

Dellaert et al investigated how Monte Carlo probabilistic models can be used to track a user more accurately as time goes on without knowing the target's initial location [49]. The approach turned out to be more effective and less memory-intensive than previous grid-based solutions.

3.1.5 Outdoor GPS Localization

There has been extensive research on the power consumption and accuracy of GPS showing the mobile GPS is both expensive in terms of power consumption and inaccurate in urban areas. Also, GPS is virtually useless in indoor settings because of the signal attenuation

caused by physical barriers. These are the reasons for research efforts attempting to find other means of indoor and outdoor localization in combination with GPS that I have previously discussed [9], [37].

3.1.6 GPS Power-cycling

GPS power consumption can greatly decrease battery life on mobile devices, like smartphones, and many researchers have investigated ways to power-cycle GPS to extend battery life while minimizing location inaccuracy. Senseless is a context-aware power-cycling approach to minimize the unnecessary use of localization sensors, like GPS, by using specific cues from less power-hungry sensors [50–52]. The researchers' experiments show that using the accelerometer to first detect user movement before activiating GPS saves battery power significantly.

Mahmoud et al. investigated how context-awareness and probabilistic models could increase the power savings of smartphone GPS power-cycling [53]. By reducing the frequency of GPS location requests from a smartphone while its user is in a previously visited area and only increasing the frequency when in a new area, the power savings can be further increased when compared to static duty cycling. The results show a 33% power saving when using their adaptive context-aware power-cycling approach versus static power cycling.

SmartDC, which is an adaptive mobility prediction-based approach to GPS power cycling, uses a mobility predictor to change the frequency at which GPS locations are requested by a smartphone device [54]. The key difference between SmartDC and other rate adaptive approaches to GPS power-cycling is the use of an energy budget shared between the process of handling location prediction and location exception. The budget allocation changes over time as the user either consistently travels on the same path to a destination or takes alternate routes. The results show that SmartDC saved an upwards of 87% more energy than other

rate-adaptive methods of GPS power cycling.

3.1.7 Seamless Localization Handoff with Technology Fusion

Many researchers have investigated methods of achieving seamless localization handoff in order to provide continuous location tracking of a target, such as a mobile robot. Kang et al researched localization fusion in their solution, called the Integrated Localization Service Framework, where multiple methods of localization, such as WiFi, RF, and GPS, can be used simultaeneously to make sure there are no dead zones when tracking mobile robots [55]. The inner workings of the framework involve using a least-squares regression model provided with data from a decision tree that toggles which active sensor data is to be currently used for localization. The motivation behind keeping all technologies active is to ensure that there are no gaps in localization handoff.

Another approach to seamless localization handoff for smartphones involves the fusion of many technologies, such as GPS, Bluetooth, WiFi, GSM/UMTS and accelerometer data [56]. The approach cycles through each technology on the smartphone and if a new piece of data is available, then it is recorded to create a Monte Carlo approach using a Particle Filter to provide seamless localization as the user moves around into new areas and areas already visited. The filter is based upon a random sampling of various types of data in the current location. Likelihood observation equations are also applied to each one of the technologies to acheive seamless localization.

3.1.8 Estimating Time of Arrival

Some researchers have investigated using smartphones as a way to estimate a user's arrival time to a destination. EasyTracker is an automatic transit tracking application for estimating arrival times of delivery trucks [57]. Each delivery truck is equipped with a smartphone that

maintains a history of arrival times at all the stops on the route and uploads the information to a shared server for other trucks to reference. As more trucks and deliveries are sent to a particular area, the more accurate the estimation becomes as the history grows.

3.2 Gesture-based End-User Authentication Related Work

Accurate methods of gesture recognition comprise a significant portion of the issues in secure smartphone pairing for collaboration. I provide a taxonomy of related work that we divide into two categories: Smartphone-based gesture recognition and non-smartphone gesture recognition.

3.2.1 Smartphone-based Gesture Recognition

Smartphones come equipped with an accelerometer that can be used in a wide range of applications. One research effort, though, defines gesture recongition in a different way through image processing [29]. Tarrataca et al. investigate hand gesture recognition using J2ME image processing on smartphones and the accuracy at which different algorithms could delineate how many fingers a user was holding up.

Niezen and Hancke investigated three different algorithmic approaches to motion gesture recognition on smartphones. They implemented approaches that utilized hidden Markov Models, artificial neural networks and dynamic time warping. Their research found that dynamic time warping was the most accurate of the three methods, showing a overall gesture recognition accuracy of 96.25%.

Joselli et al. researched both motion and touch gestures on smartphones as a feasibility study of different types of UI for gaming purposes [58]. gRMobile is a Hidden Markov Model

algorithm that characterizes and recognizes various touch and motion gestures using the device's touchscreen and accelerometer, respectively.

3.2.2 Non-smartphone Gesture Recognition

Some research efforts have explored gesture recognition on electronic devices other than smartphones. One example is the Nintendo Wiimote, which has a highly accurate accelerometer built into the device [27]. uWave is a gesture recognition algorithm that can be used on mobile devices that have an acclerometer for UI and gaming purposes. The approach uses data quantization and time warping to process incoming accelerometer data.

Gesture Watch is another non-smartphone gesture recognition research effort that focused on using a watch worn on the user's wrist to monitor gestures for communication purposes [28]. With the device, a user can control other electronic devices, such as media players, to play and pause music. Gesture watch achieves gesture recognition through the use of five infrared proximity sensors.

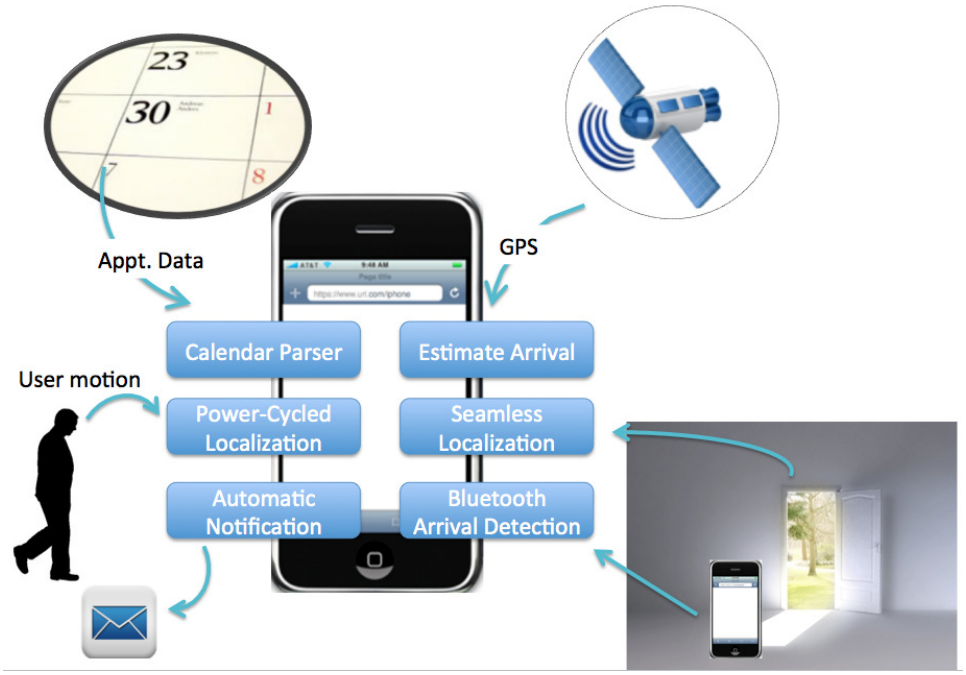
Chapter 4

Overview of PENDING

4.1 User Late Arrival Detection Algorithm on Smartphones with PENDING

To provide a solution in the user late arrival detection problem domain, I developed a smartphone application, called PENDING, that acheives cyber-physical power-efficient user late arrival detection. Figure 4.1 is a high-level diagram of PENDING's features and solution components. PENDING consists of calendar data extraction, motion detection, estimated time of arrival calculation, automatic notifications, seamless indoor/outdoor localization handoff, and arrival detection with indoor localization. PENDING's features address the four challenges described in Section 2.3: (1) low power consumption with an accelerometer-based algorithm to determine when to start localization; (2) relative indoor localization without extra preconfigured hardware beyond what is available on the smartphone; (3) seamless indoor/outdoor localization handoff with a GPS-based algorithm; and (4) estimating arrival time without location fix. These challenges are overcome by PENDING through the use of four critical techniques that are driven by physical environment inducement for algorithm

Figure 4.1: PENDING's components and features.



adaptation: (1) localization sensor power-cycling; (2) seamless indoor/outdoor localization; (3) context-aware automatic notification; and (4) user-assisted cyber model memory.

4.1.1 PENDING's Formal Model

Late arrival detection requires the processing of physical stimulus data from a smartphone's sensors, estimation of a user's location/context, and determination of whether a user will reach a meeting location by its start time. PENDING's late arrival detection algorithm is based on a 6-tuple state representation that encodes physical stimulus data and location/-context estimation:

$$Ls_k =< t_k, D(t_k), C(t_i), \phi(t_i), \rho(t_i), \sigma(t_i) > \tag{4.1}$$

where:

- $t_k \subset T$ is the time of the future scheduled meeting M_k, from the user's calendar. The set of all future meeting times is T.
- $D(t_k)$ is the latitude and longitude of the meeting, M_k, at time t_k.
- $C(t_i)$ is the last known latitude and longitude of the device at time t_i, where $t_i < t_k$.
- $\phi(t_i)$ is a flag indicating whether or not the device is believed to be indoors or outdoors at time t_i. If the device is indoors, $\phi(t_i) = 1$.
- $\rho(t_i)$ is a flag indicating whether or not the device is believed to be currently moving. If the device is believed to be moving, $\rho(t_i) = 1$.
- $\sigma(t_i)$ is a flag indicating whether or not the device has received a Bluetooth beacon signal from the device of a user who has arrived at the meeting location. If the signal has been received, $\sigma(t_i) = 1$.

Late arrival detection is modeled as a function, $Late(Ls)$, where the determination of whether or not a user is late is based on the device's current location, the estimated time to reach a future meeting location, and whether or not a Bluetooth beacon signal has been received for the given meeting:

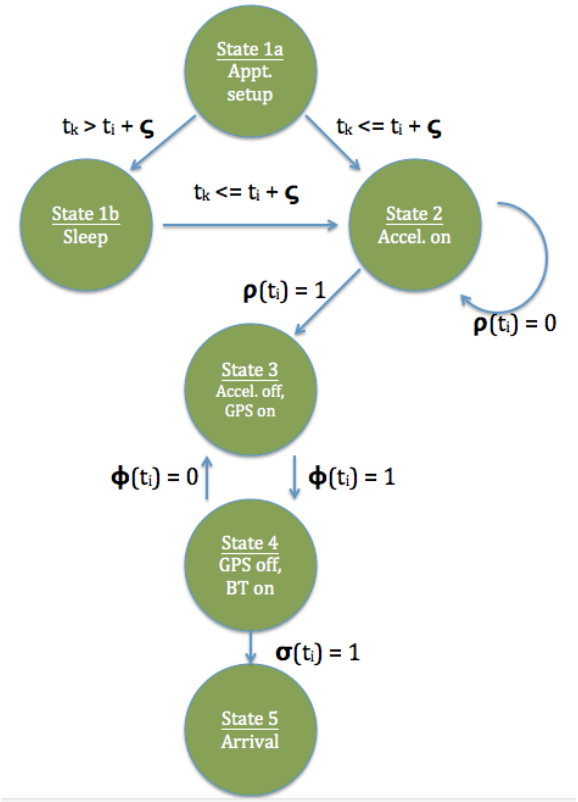
$$L(M_k) = \begin{cases} t_i + \Delta(D(t_k), C(t_i)) \leq t_k + \zeta \wedge \alpha = 0 & 0\ (a), \\ t_i + \Delta(D(t_k), C(t_i)) + \epsilon \leq t_k + \zeta & 0\ (b), \\ \sigma(t_i) = 1 & 0\ (c) \\ t_i + \Delta(D(t_k), C(t_i)) + \zeta > t_k & 1\ (d), \\ otherwise & 1\ (e) \end{cases} \tag{4.2}$$

where:

- M_k is a future meeting.
- t_i is the current time on the user's device.
- $\Delta(D(t_k), C(t_i))$ that estimates the travel time from the user's current last known location to the meeting location.
- ζ is a lateness threshold factor that determines how many minutes past a meeting's start time the user can arrive without being considered late. This factor can be varied based on meeting importance, local customs, and other factors.
- α is a flag that indicates whether or not arriving at the meeting destination is sufficient to have arrived at the meeting or if the attendance requires reaching a specific room.
- ϵ is a time allowance to walk from the entrance of a building to a meeting room. I used a fixed value for ϵ. In practice, ϵ will vary based on the building, however we assume that in most cases ϵ is negligible and will not make a substantial impact. If needed, this value could be dynamically assigned based on the building and room that the meeting is scheduled in.

The PENDING late arrival detection algorithm considers a user to be on-time in three primary cases. In case (a), the meeting is not taking place indoors or within a specific room ($\alpha = 0$) and the current time plus the travel time from the user's current location to the meeting location is less than the meeting start time of the meeting (t_k). Case (b) focuses on situations in which the user has not reached the destination location yet and the meeting is occurring in a specific room. In this situation, if the user can reach the destination location and walk to the room before the meeting start time, the user is considered on-time. The

Figure 4.2: PENDING's state-machine diagram.

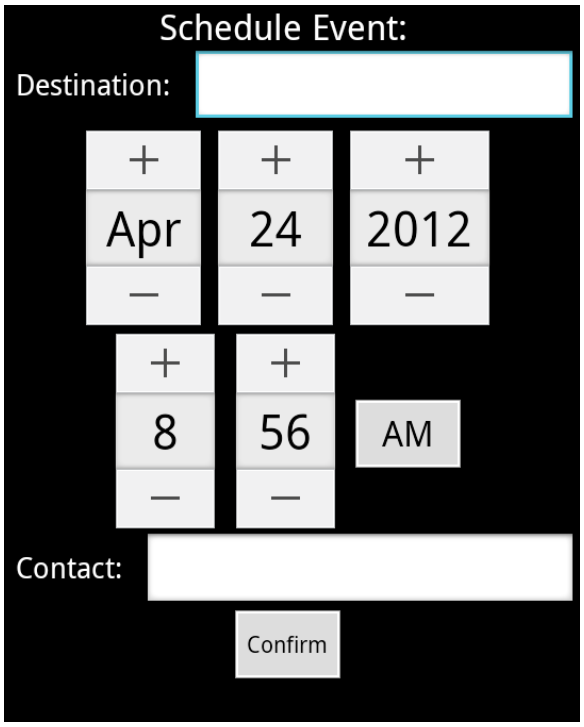


final on-time case, case (c), marks the user as being on-time because a Bluetooth beacon signal for the meeting has been received from another attendant's smartphone. In all other cases, the user is expected to arrive late to the meeting.

4.1.2 PENDING's State Model

Physical events and changes in the user's environment provide the inputs that drives the algorithm. The algorithm can be represented by a state-machine where each state represents a specific smartphone sensor that is active. The state-machine can be seen in Figure 4.2. The state-machine has five states: (1) appointment setup/application sleep, (2) accelerometer enabled, (3) GPS enabled, (4) Bluetooth enabled and (5) destination reached.

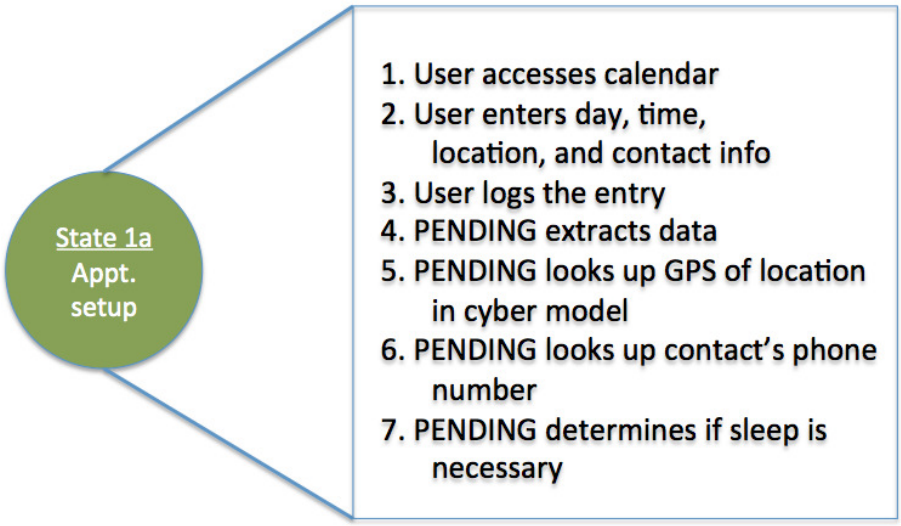
Figure 4.3: Scheduling widget.



State 1a: Appointment setup. State 1a involves the start up of the application and the user scheduling an appointment in the phone by use of a calendar widget, such as the prototype calendar widget used for our experiments and shown in Figure 4.3.

The full list of steps that occur withing State 1a can be seen in Figure 4.4. Once the user selects the time of the appointment, the destination, and the name of a contact that will be present at the appointment, the meeting data is stored for later use by the decision tree in the cyber model. The geolocation data of the destination is extracted by referencing a look-up table of location name and GPS coordinate pairs. If the name of the destination matches one of the pairs listed in the look-up table, then the associated GPS coordinates are stored in the cyber-model. I assumed that the appointments are at destinations on the Virginia Tech campus, but we do not assume that the cyber model has all GPS coordinates for every building stored in the look-up table. The scenario where the GPS coordinates of

Figure 4.4: The steps that occur during State 1a.

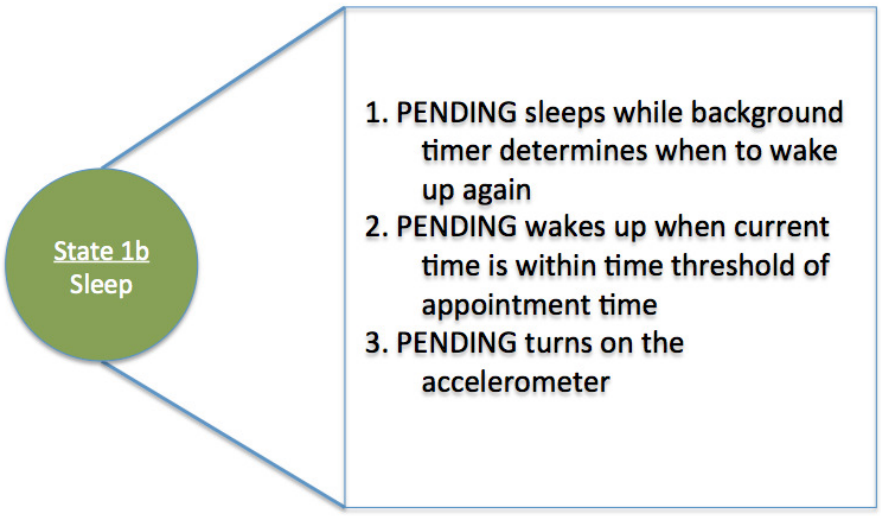


the destination are unknown is discussed later when we explain the user-assisted cyber-model memory feature of PENDING.

State 1b: Sleep. The full list of steps that occur during State 1b can be seen in Figure 4.5. If PENDING determines that the application must sleep, then the main thread goes to sleep until the current time is within γ minutes of the appointment time selected in State 1a. For example, if the user scheduled an appointment for 11:30 AM, $\gamma = 5min$, and the current time is 11:05 AM, then the application would sleep for twenty minutes and then wake back up at 11:25 AM. In our experiments, a buffer time of five minutes was shown to effectively detect late arrivals in most cases without excessively consuming battery power.

State 2: Accelerometer on. Figure 4.6 shows the full list of steps that occur during Step 2. An important energy saving step of PENDING is that it attempts to only localize the user if movement is detected. PENDING uses physical motion as an input to adapt the localization algorithm in order to save power. The PENDING formal model captures

Figure 4.5: The steps that occur during State 1b.

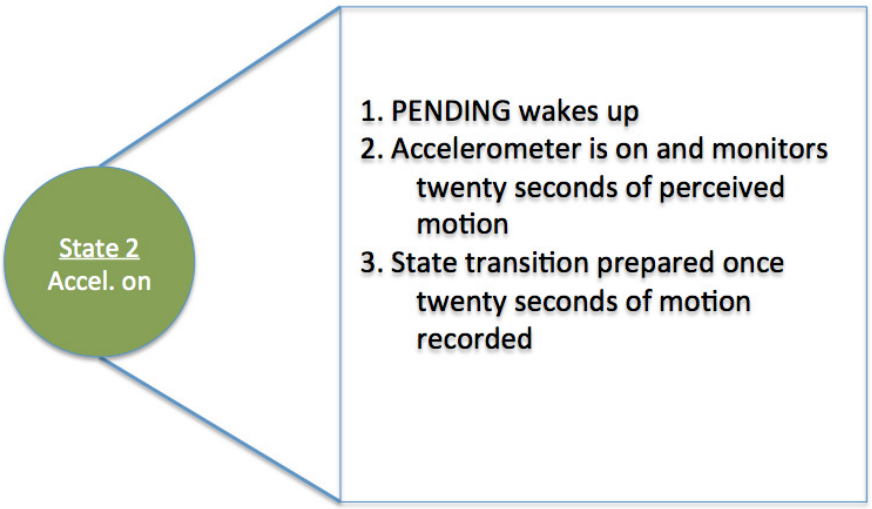


perceived motion in $\rho(t_i)$. A smartphone's accelerometer is used to calculate $\rho(t_i)$.

In State 2, the application is woken up and the smartphone's accelerometer is enabled to record physical acceleration experienced by the phone caused by the user's movement. For example, if the user is carrying the phone as he/she walks around, the accelerometer will record the changes in acceleration in the x, y, and z-direction. This physical data is used by the decision tree to enable the localization sensors. PENDING uses a threshold function based on the total movement recorded over a configurable duration, β. If the recorded motion exceeds a threshold, the smartphone is considered to be in motion and $\rho(t_i) = 1$. Otherwise, $\rho(t_i) = 0$.

State 3: Accelerometer off/GPS on. The steps that comprise State 3 can be seen in Figure 4.7. At regular intervals, PENDING must attempt to determine the device's current location and update $C(t_i)$. PENDING uses a smartphone's GPS sensor to derive a location for $C(t_i)$. GPS is also used to make a determination of whether or not the device is indoors

Figure 4.6: The steps that occur during State 2.

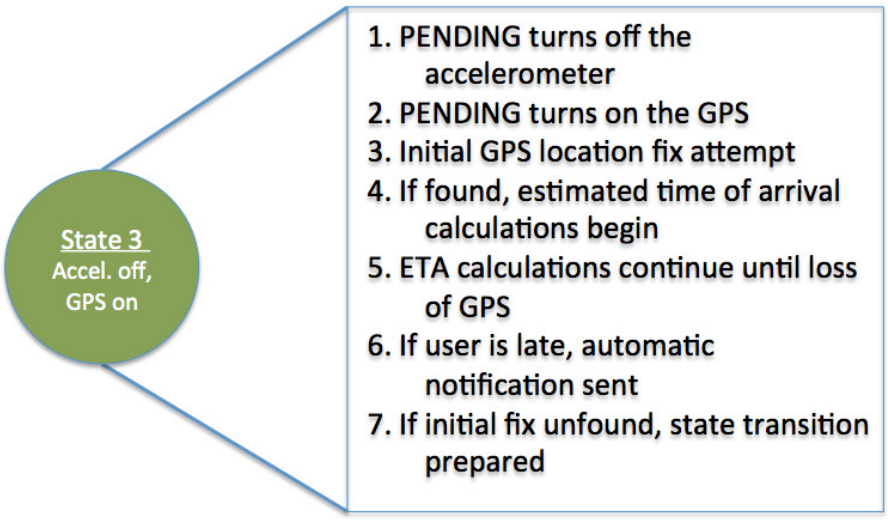


and assign $\phi(t_i)$.

In State 3, the accelerometer is disabled and acceleration data is no longer recorded since the decision tree has decided that the user is moving around. GPS is enabled on the smartphone to establish an initial outdoor location fix. Once a location fix is established, location updates are sent to the smartphone periodically as the user walks to update $C(t_i)$. When a new location update is sent to the phone, a background count down timer is reset and initiated by the decision tree. If this timer reaches zero, then the decision tree determines that the user is no longer outside and has entered a building ($\phi(t_i) = 1$).

State 3: Estimated time of arrival While GPS is on and if the GPS coordinates of the destination are known, estimated time of arrival calculations are done by calculating the user's current speed and determining how long it will take to reach the destination at that speed. I used the haversine formula, which compensates for the curvature of the earth, to calculate the great-circle distances between two consecutive user location updates. The current speed is calculated by dividing the distance difference with the time elapsed between

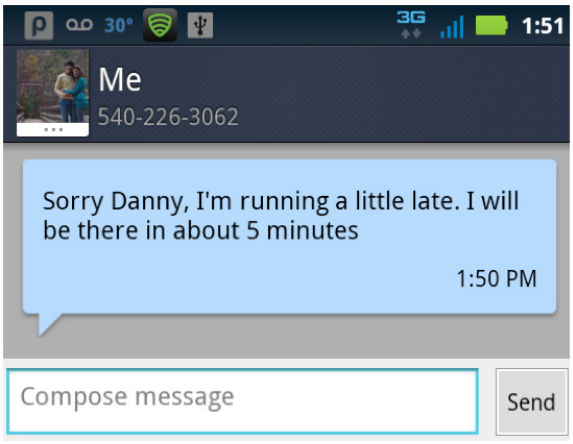
Figure 4.7: The steps that occur during State 3.



the two consecutive location updates. With the known difference in distance between the user and the destination and the user's current speed, the adapting algorithm figures out the estimated time of arrival.

State 3: Automatic notification Once the user's estimated time of arrival is calculated, the cyber-model determines if there is enough time unti the start of the event for the user to be able to arrive at the destination on time, represented by $\Delta(D(t_k), C(t_i))$. If the cyber-model determines that the user's arrival time extends past the lateness threshold, ζ, then an automatic notification is prepared. Recall that in State 1a the name of a contact who is to be present at the appointment is stored in the cyber-model. The algorithm searches the smartphone's contact list for a match in name and then extracts the contact's cell phone number or e-mail address; priority given to the cell phone number since automatic SMS delivery requires no user interaction, whereas an automatic e-mail still requires the user's selection of an e-mail client installed on the smartphon before delivery. The automatic notification feature then automatically sends an SMS or e-mail to the contact notifying him

Figure 4.8: Automatic text notification.

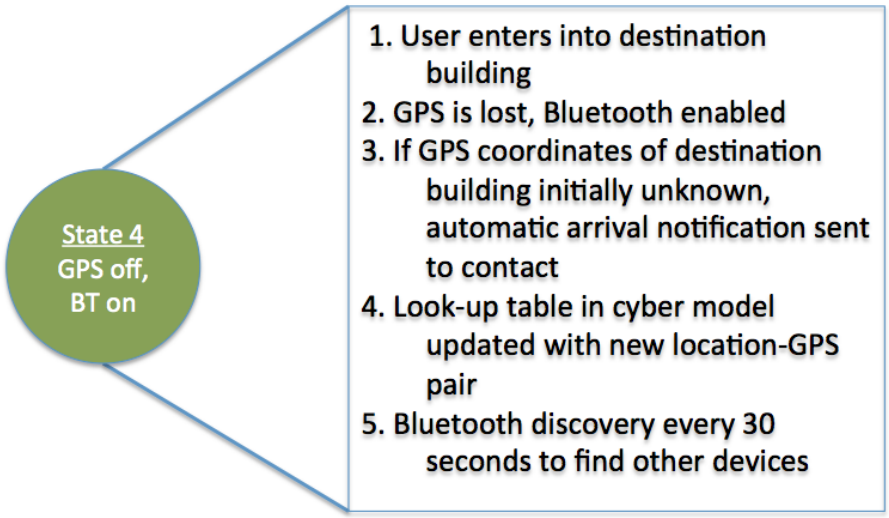


or her the amount of time it will take for the user to reach the destination. Figure 4.8 show the automatic text notifications prepared by the cyber-model.

State 4: GPS off/Bluetooth on. See Figure 4.9 for the full list of steps that occur during State 4. For indoor meeting locations, a Bluetooth rendez-vous scheme is used to detect when a user has reached the meeting location. The first meeting participant to enter the meeting location enables a Bluetooth beacon on their device that broadcasts the unique ID of the meeting. When other devices discover this Bluetooth beacon, they update $\sigma(t_i)$ to 1. The function $\sigma(t_i)$ serves as an indicator that the user has been localized to the meeting room.

State 4 begins the Bluetooth discovery process to update $\sigma(t_i)$. This state is reached when the phone no longer receives location updates causing the count down timer from State 3 to reach zero. The smartphone's GPS is disabled and the Bluetooth adapter is turned on. The adapting algorithm uses the environment shift from outdoor to indoor to begin the indoor localization process. The adapter is put into discovery mode to detect the meeting's Bluetooth beacon.

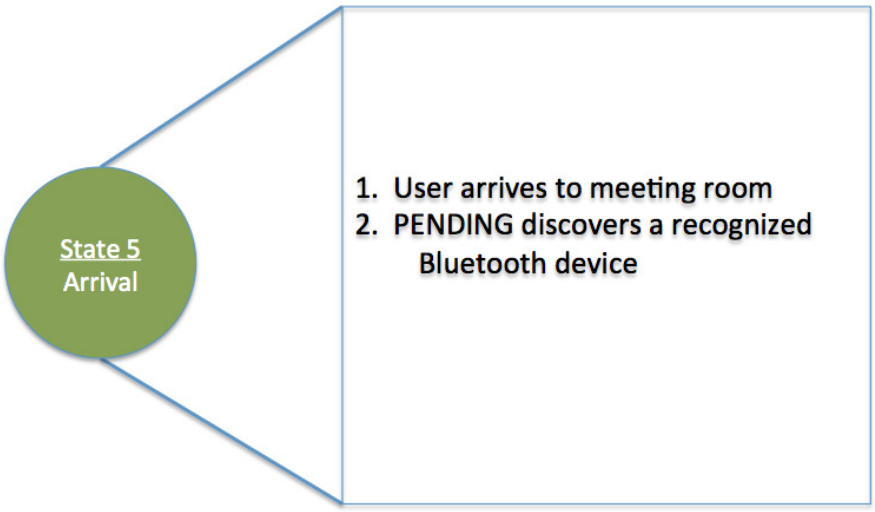
Figure 4.9: The steps that occur during State 4.



State 4: User-assisted cyber-model memory Recall in State 1a that if the destination can be found in the look-up table of destination-GPS pairs, then those GPS coordinates are stored in the cyber-model for estimating the user's arrival time. In the event that the destination cannot be found in the look-up table, the estimated time of arrival and automatic notification becomes unavailable. Once the environment shift from outdoor to indoor occurs, the adapting algorithm then prompts the user if he or she has arrived to the destination. If yes, the cyber-model saves the last known GPS coordinates of the user as the location data for that destination for future events. The look-up table is then updated to reflect the addition of the new data.

State 4: Context-aware automatic notification Recall that an automatic notification is sent during State 3 if $\Delta(D(t_k), C(t_i))$ is greater than ζ. The automatic notification notifies the contact how many minutes the user is away from arriving at the destination. In the scenario where the destination's GPS coordinates are unknown in State 1a , automatically notifying the contact that the user has arrived at the entrance of the building would be useful.

Figure 4.10: The steps that occur during State 5.



Again, the environmental shift from outdoors to indoors induces PENDING's algorithm adaptation to send a notification message regardless of whether or not the GPS coordinates of the destination were known from the start.

State 5: Devices found. Figure 4.10 shows the steps that occur in State 5. In State 5, if the correct Bluetooth broadcast is discovered in the user's environment, then the user is at the destination.

4.1.3 Power Consumption and Localization

Context-based Power Cycling. As described in Section 2.3.1, power efficiency is a major challenge in smartphone-based late arrival detection.

In order to conserve battery power, PENDING uses a context-based approach to power GPS and Bluetooth sensors on and off. After numerous isolated tests of turning localization sensors on and off, two key discoveries were made. The first key discovery was that there

was a specific window of time that the accelerometer needed to be on in order to maximize power savings and reduce both the false interpretation of the user's movement as well as the loss of accuracy in establishing an initial location fix. If the window of time was too short, and $\rho(t_i)$ was set to 1 too early, then the user's unintentional movements was interpreted by the cyber-model as the user heading towards the destination and more power was wasted. If the window of time was too long, and $\rho(t_i)$ was set to 1 too late, more power was saved but the algorithm ran the risk of activating GPS too late once the user was already inside the destination. The observance of environmental shifts, $\phi(t_i)$, that induce the algorithm adaptation are lost in both scenarios, thus making this component of PENDING especially important. I observed that a time window of twenty seconds provided the best results in reducing the occurance of the two scenarios.

The second key discovery in conserving battery power with power-cycling involves the frequency at which Bluetooth discoveries are conducted when performing indoor localization at the destination. If the frequency of Bluetooth discoveries done by the smartphone were more frequent, the discovery of the destination beacon came sooner but the power trade-off made the option costly to battery life. If the frequency of Bluetooth discoveries were less frequent, then the power consumption was much less, but the time at which the destination beacon was discovered was not indicative to the time at which the user actually arrived at the destination. I observed that a Bluetooth discovery frequency of once every thirty seconds provided the best results.

textbfIndoor Localization Approach. Once the user is indoors, GPS-based localization is no longer effective due to signal attentuation caused by walls. Adding extra equipment designed for indoor localization is problematic and costly and dead reckoning isn't accurate enough. Indoor meetings require a smartphone to localize a user to a specific room to determine if the user has entered the meeting room. However, when a user is inside the building containing

a meeting room, it is typically not necessary to know exactly which room they are in. Once a user is in the building where a meeting is taking place, knowing whether the user is in the designated meeting room or not is sufficient. In most cases, the travel time within a building is insignificant.

Because it is not necessary to know exactly where a user is within the building of a meeting location, PENDING uses a simple Bluetooth rendez-vous beacon to determine when a user has reached a meeting room. When the first participant in a meeting reaches the designated meeting room, they activate a Bluetooth beacon that broadcasts a unique ID assigned to each meeting. When the devices of other participants in the meeting receive the Bluetooth broadcast, they automatically assume that they have reached the meeting room. This localization approach is similar to prior Bluetooth-based indoor localization approaches, but does not require the installation of external hardware. The approach relies on the short range of Bluetooth communications to determine if a user has reached a meeting room.

Chapter 5

PENDING Test Plan and Results

5.1 PENDING's Experimental Platform

I tested PENDING's functionality by running real-world tests with the application. I was interested in several results: (1) how much power PENDING's localization feature saved in comparison to other localization approaches, (2) how accurately PENDING could detect late arrivals and send approrpriate automatic notifications, (3) how seamlessly PENDING switches between indoor and outdoor localization and (4) how consistently PENDING responds to non-ideal scenarios, such as new destinations and loss of location fix.

The evaluation metrics were: (1) PENDING's battery use during localization compared to the battery use of other localization approaches, (2) the success rate at which PENDING appropriately sent automatic notifications when the user was late, (3) the success rate at which PENDING correctly interpreted and outdoor to indoor environmental shift and (4) the consistency at which PENDING asked the user for destination arrival confirmation when the location data was initially unavailable and the success rate at which PENDING automatically

sent a notification to the contact when the user moved from outdoors to indoors.

I began real-world tests with the application installed on a Droid smartphone with Android version 2.2.2. The testing schedule was set up to parallel the state transitions described in Section 4.1.2. The user begins by inputing a new appointment entry into PENDING's calendar widget. User lateness detection begins once the user saves the appointment to the cyber-model. PENDING's main thread sleeps until the current time is within the threshold of five minutes of the appointment time, then the accelerometer is activated.

I designed and ran five experiments for PENDING to test its performance and power-efficiency. The first experiment tested PENDING's ability to predict late arrivals. The second experiment compared PENDING's power consumption to that of late arrival detection with purely GPS-based localization. The third experiment compared PENDING's power consumption to that of late arrival detection with purely Bluetooth-based localization. The fourth experiment tested PENDING's estimated time of arrival calculations and automatic notification features. The fifth experiment tested PENDING's ability to update the cyber model in scenarios where the geolocation data of the destination was not initially known.

5.1.1 Predicting Late Arrivals Experiment

The first experiment was designed to test PENDING's ability to detect late arrivals and automatically notify the contact at the desination of the user's lateness. The experiment tested at a holistic level to prove the concept of physical environment induction for algorithm adaptation. I specified four case scenarios for the experiment: (1) user always indoors, (2) user travels from outdoor to indoor, (3) user starts indoors, then travels outdoors and finally arrives to an indoor desintation and (4) the user is always outdoors. Additionally, we specify expected checkpoints, such as successful localization handoff and successful automatic noti-

Figure 5.1: An example table of results for the predicting late arrivals experiment.

Case #1 (Late)		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall			N/A	
	Cassell to Wallace Hall			N/A	
	Lee Hall to Patton Hall			N/A	
	Dietrich to Main Eggleston			N/A	
	War Memorial to Lee Hall			N/A	

Case #1 (On Time)		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall		N/A	N/A	
	Cassell to Wallace Hall		N/A	N/A	
	Lee Hall to Patton Hall		N/A	N/A	
	Dietrich to Main Eggleston		N/A	N/A	
	War Memorial to Lee Hall		N/A	N/A	

fication, for each of the case scenarios that PENDING must meet in order for the algorithm to be able to effectively detect user lateness. Each case was tested on one of five different paths that the user could possibly take on the Virginia Tech campus.

A table of results for each trial, as seen in Figure 5.1, was recorded for later analysis. The "Contact Info" field specifies whether or not the cell phone number and e-mail address of the contact of the destination was obtained, the "Motion" field specifies whether or not PENDING detected user motion during the run, the "Initial GPS" field means whether or not PENDING was able to obtain an initial location fix on the user. The "ETA" field specifies whether or not an estimated time of arrival value was calculated during the test run, the "Notify" field on the table of results means if PENDING sent an automatic notification to the contact at the destination, the "Handoff" field designates whether or not successful localization handoff's took place in scenarios where the user's environment changed while en route to the destination. Finally, the "Arrival" field means whether or not PENDING was able to determine the user's eventual arrival to the destination.

Each table of result for each case was slightly different in that certain checkpoints were greyed out because they were not applicable to the scenario. For example, in Figure 5.1 we see that in "Case 1 (Late)" the "Initial GPS" and "Handoff" fields are greyed out because Case 1 is where the user is always indoors. Thus, an initial GPS fix is't necessary and there is not localization handoff because the user's environment doesn't change in terms of indoors or outdoors. Note in "Case 1 (On Time)", the "Notify" field is also greyed out because the user is never late in that scenario, thus PENDING does not need to send an automatic notification.

Before running the experiment, we made some base assumptions. The first assumption was that there would always be a contact at the destination who would have his smartphone on his person with Bluetooth enabled and ready to receive automatic notifications in the form of SMS messages or e-mails. The second assumption was that the target user would continually be in motion once he began heading to the destination. It was not pertinent for PENDING to have the ability to determine if the user was moving throughout the entire test run. Finally, we assumed that all destinations could be reached by walking, so no driving scenarios were considered, even though PENDING's algorithm would not be greatly affected by difference modes of user transportation.

Hypothesis: PENDING's Late Arrival Detection Varies by Scenario My hypothesis was that PENDING's accuracy in detecting late arrivals would vary case by case. Specifically, we predicted that the case where the user was always outside would yield the most favorable results in terms of PENDING's ability to detect whether or not the user was running late. As described before, PENDING uses physical environment induction for algorithm adaptation, so any change in the environment or other physical trigger will provoke a change in how the algorithm detects the user's lateness. I predicted that these changes would have a minute affect on PENDING's accuracy.

Figure 5.2: Table of results for case 1, user always indoors.

Case #1 (Late) - User Indoors		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall	Not Met	Met	N/A	Met
	Cassell to Wallace Hall	Not Met	Met	N/A	Met
	Lee Hall to Patton Hall	Not Met	Met	N/A	Met
	Dietrich to Main Eggleston	Not Met	Met	N/A	Met
	War Memorial to Lee Hall	Not Met	Met	N/A	Met
Case #1 (On Time)		**ETA?**	**Notify?**	**Handoff?**	**Arrival?**
Path:	Squires to Owens Hall	Not Met	N/A	N/A	Met
	Cassell to Wallace Hall	Not Met	N/A	N/A	Met
	Lee Hall to Patton Hall	Not Met	N/A	N/A	Met
	Dietrich to Main Eggleston	Not Met	N/A	N/A	Met
	War Memorial to Lee Hall	Not Met	N/A	N/A	Met

PENDING's User Lateness Detection Results I ran a total of 40 test runs to test PENDING's user lateness detection in four different cases. Each case had two user conditions, late and on time, and each user condition had five destinations as seen in Figure 5.1.

Below are the results for each of the four cases:

Figure 5.2 shows PENDING's performance in case 1 where the target user was always indoors. There were consistent results in each of the trials in terms of which checkpoints were met and which weren't. PENDING was able to obtain the contact data, detection motion, notify the contact after determining that the user was late because the user wasn't at the destination at the appointment time and detect when the user finally made it to the destination. PENDING was unable to calculate and estimated time of arrival because outdoor GPS is necessary to do so, which wasn't available in the indoor only case.

Figure 5.3 shows PENDING's performance in case 2 where the target user starts outdoors and then travels to a destination that is indoors. As in case 1, the user lateness detection

Figure 5.3: Table of results for case 2, user travels from outdoors to indoors.

Case #2 (Late) - Out to In		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall	Met	Met	Met	Met
	Cassell to Wallace Hall	Met	Met	Met	Met
	Lee Hall to Patton Hall	Met	Met	Met	Met
	Dietrich to Main Eggleston	Met	Met	Met	Met
	War Memorial to Lee Hall	Met	Met	Met	Met

Case #2 (On Time)		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall	Met	N/A	Met	Met
	Cassell to Wallace Hall	Met	N/A	Met	Met
	Lee Hall to Patton Hall	Met	N/A	Met	Met
	Dietrich to Main Eggleston	Met	N/A	Met	Met
	War Memorial to Lee Hall	Met	N/A	Met	Met

algorithm effectively detected all the scenarios where the user was running late and notified the contact at the destination. The algorithm also detected that the user was not late in the test runs where the user was on time, so automatic notifications were not sent.

Figure 5.4 shows PENDING's performance in case 3 where the target user starts indoors, then travels outdoors and then reaches an indoor destination. In comparison to the other cases, case 3 has more opportunities for changes in the user's environment. Each scenario had two localization handoffs, one for the indoor to outdoor transition and another for the outdoor to indoor destination transition. In general, PENDING was able to detect user lateness as the environment shifted from indoors to oudoors and vice versa. There was one instance where PENDING did not detect the user's lateness and one instance where PENDING calculated the user as late even though the user was on time to the destination. Possible causes of these differences may lie in the complexity of the user's path and time spent in each environment.

Figure 5.4: Table of results for case 3, user travels from indoors to outdoors then back indoors.

Case #3 (Late) - In->Out->In		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall	Met	Met	Met	Met
	Cassell to Wallace Hall	Met	Met	Met	Met
	Lee Hall to Patton Hall	Met	Met	Met	Met
	Dietrich to Main Eggleston	Met	Met	Met	Met
	War Memorial to Lee Hall	Met	False Positive	Met	Met

Case #3 (On Time)		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall	Met	N/A	Met	Met
	Cassell to Wallace Hall	Met	N/A	Met	Met
	Lee Hall to Patton Hall	Met	N/A	Met	Met
	Dietrich to Main Eggleston	Met	False Negative	Met	Met
	War Memorial to Lee Hall	Met	N/A	Met	Met

Figure 5.5: Table of results for case 4, user always outdoors.

Case #4 (Late) - User Outdoor		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall	Met	Met	N/A	Met
	Cassell to Wallace Hall	Met	Met	N/A	Met
	Lee Hall to Patton Hall	Met	Met	N/A	Met
	Dietrich to Main Eggleston	Met	Met	N/A	Met
	War Memorial to Lee Hall	Met	Met	N/A	Met

Case #4 (On Time)		ETA?	Notify?	Handoff?	Arrival?
Path:	Squires to Owens Hall	Met	N/A	N/A	Met
	Cassell to Wallace Hall	Met	N/A	N/A	Met
	Lee Hall to Patton Hall	Met	N/A	N/A	Met
	Dietrich to Main Eggleston	Met	N/A	N/A	Met
	War Memorial to Lee Hall	Met	N/A	N/A	Met

Figure 5.5 shows PENDING's performance in case 4 where the target user is always otudoors. As expected, PENDING was able to accurately detect when the user was running late and send an automatic notification. Since the user was always outdoors, an estimated time of arrival was able to be calculated since outdoor GPS was available.

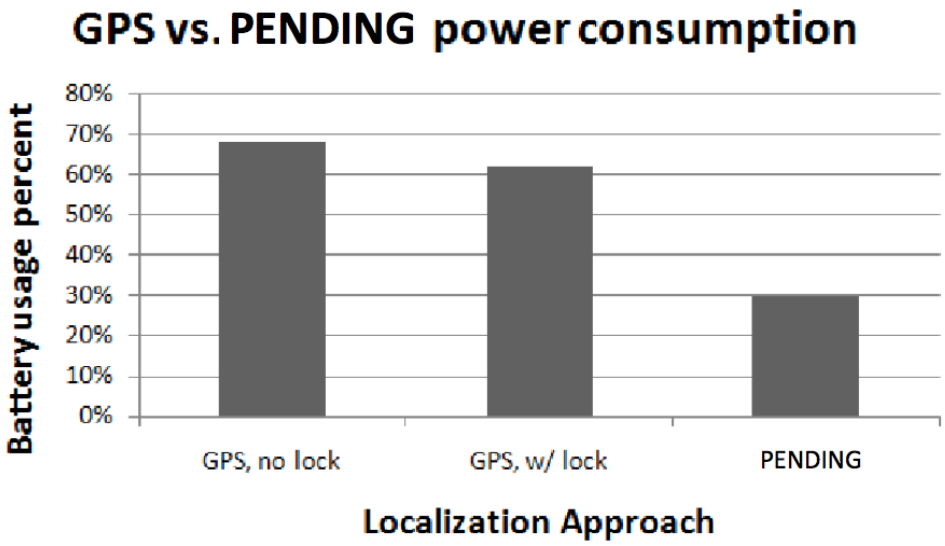
5.1.2 Pure GPS vs. PENDING Power Experiment

The second experiment was designed to compare PENDING's battery usage on the smartphone during localization versus the battery usage of GPS-based localization on a smartphone. The GPS-based localization approach enables the smartphone's GPS and continuously displays the GPS coordinates of the user to the screen. To monitor battery consumption, we used a battery monitoring widget available on the Android market that shows the battery consumption of each running process as a percentage of total power allocation. Typical processes that contribute to battery usage of the smartphone include the backlight for the display, call standby and WiFi.

Before running our application and the GPS application, we rebooted the phone to clear all battery usage history to establish a level playing field. Each solution ran for six minutes, which was the typical amount of time needed for the user to reach the destination. I conducted this experiment for six trials to obtain an average power consumption value for each application.

Hypothesis: Motion-based power cycling is more power-efficient than continually using GPS. My hypothesis was that running GPS outside would have a higher battery usage percentage than PENDING and running GPS without acquiring an initial position lock by staying indoors would yield an even higher percentage than both outdoor GPS and PENDING. It was assumed that the constant overhead in attempting to establish a lock would consume more power than walking around outside with periodic location updates.

Figure 5.6: Battery usage of PENDING vs. purely GPS-based late arrival detection.

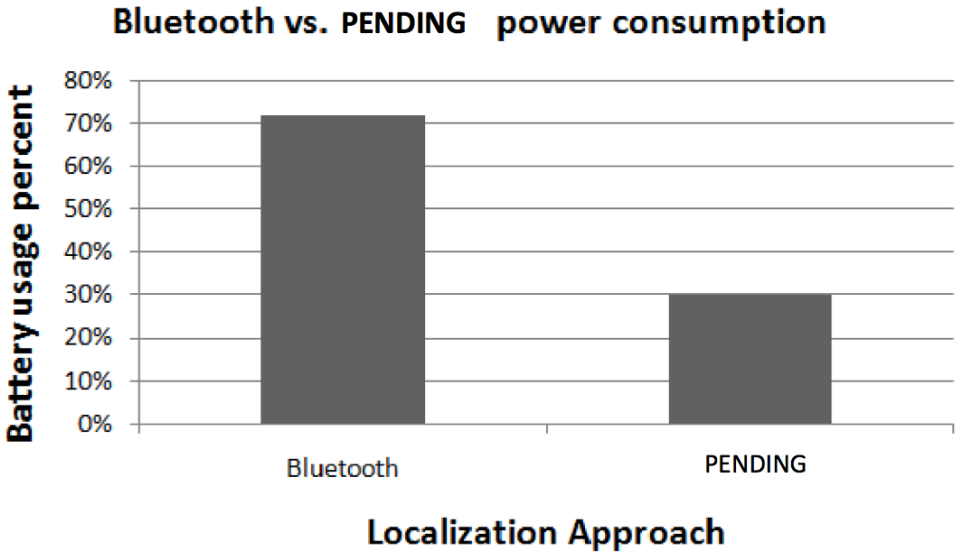


Pure GPS Late Arrival Detection vs. PENDING Power Consumption Results. As seen in Figure 5.6, pure GPS-based localization active for six minutes consumed more power than six minutes of PENDING's localization. Running the GPS without an initial position lock constituted 68% of the smartphone's battery use and GPS with position lock was slightly lower at 62%. These figures are more than double the 30% battery usage level of PENDING.

5.1.3 Pure Bluetooth vs. PENDING Power Experiment

The third experiment was designed to determine whether pure Bluetooth-based localization consumed more power than PENDING's Bluetooth-based indoor localization. When the smartphone's Bluetooth adapter is put into discovery mode, it sends out discovery requests for a default time of twelve seconds to find nearby Bluetooth devices that the requesting smartphone can pair and connect with. The pure Bluetooth-based localization approach was written to keep the smartphone device in constant Bluetooth discovery mode, like some

Figure 5.7: Battery usage of PENDING vs. purely Bluetooth-based late arrival detection.



indoor localization approaches researched before. I conducted this experiment for six trials to obtain an average power consumption value for each application.

Hypothesis: Continual Bluetooth localization consumes more power than power-cycled localization. Ourhypothesis was that pure Bluetooth indoor localization would consume more battery than PENDING's indoor localization. I also expected to observe pure Bluetooth power consumption even higher than that of the pur GPS localization solution because of the comparatively more overhead involved in Bluetooth discovery.

Pure Bluetooth Late Arrival Detection vs. PENDING Power Consumption Results.

As seen in Figure 5.7, the pure Bluetooth-based indoor localization approach consumed more battery power during the six-minute test than both the GPS-based localization approach we PENDING.

5.1.4 Estimating Arrival Time and Automatic Notification

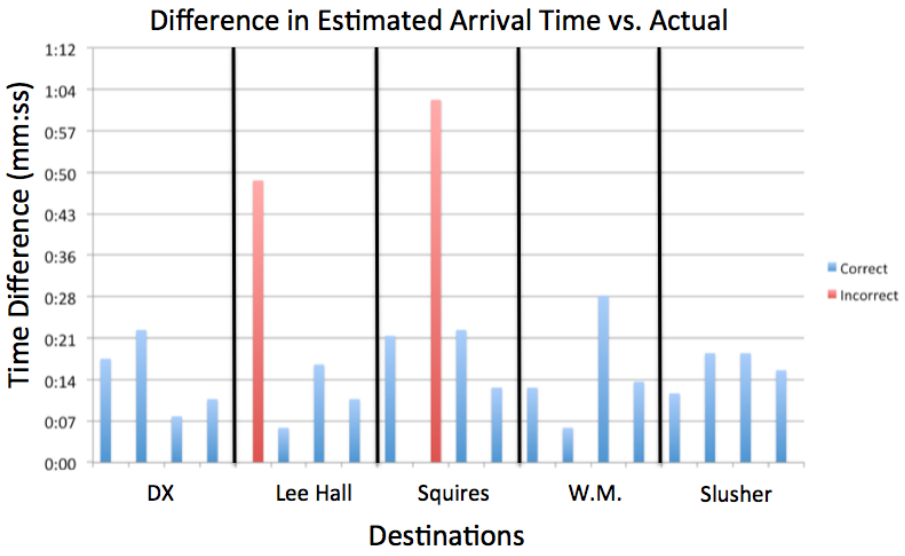
The fourth experiment was designed to independently evaluate PENDING's accuracy in calculating estimated arrival times and automative notification feature while the user was late to an appointment. Five different buildings on the Virginia Tech campus were chosen where the user could schedule appointments and move to. The starting location was always the same for each trial. I ran four trials for each location, and kept track of how close PENDING's estimated arrival time was to the actual user arrival time. In this experiment, we used destinations whose location data had previously been saved into the look-up table. I also kept track of how appropriately PENDING sent automatic notifications.

Hypothesis: PENDING has accurate ETA calculations and can send automatic notifications My hypothesis for this experiment was that PENDING would be accurate enough to automatically prepare and send a text message or e-mail after estimating the user's arrival time. It was also predicted that PENDING would not send automatic notifications if the user was not running late.

Estimating User Arrival Time and Automatic Notification Results

As seen in Figure 5.8, PENDING was able to correctly detect user late arrivals 90% of the time with an average difference in time of 0:19 seconds between the actual arrival time to the building and the estimated user arrival time, a standard deviation of 0:13.47 seconds and 95% confidence interval of 0:19 +/- 0:5.21 seconds. Additionally, PENDING sent an automatic SMS to each contact whenever the algorithm detected that the user was running late.

Figure 5.8: Estimated arrival time vs. actual arrival time.



5.1.5 Seamless Localization Handoff

The fifth experiment was designed to independently evaluate PENDING's ability to seamlessly handoff outdoor-to-indoor localization. The experiment was also designed to test our solution's ability to recognize a new destination and appropritely prompt the user for confirmation upon arrival. Using the same five buildings and three location-unknown buildings, we ran four tests for each building to see if PENDING was able to seamless toggle between indoor and outdoor localization and update the cyber model's look-up table when entering a new building for the first time.

Hypothesis: PENDING possesses seamless localization handoff and appropriately updates the cyber model I predicted that PENDING would be able to seamlessly switch between outdoor and indoor localization whenever the user moved from an outdoor environment to an indoor environment. I also predicted that PENDING would be able to consistently recognize new destinations and ask the user to confirm location data extraction

once GPS is inactive.

Seamless Indoor and Outdoor Localization Handoff and New Destination Handling Results I observed that PENDING was able to seamlessly switch from outdoor localization to indoor localization every time the user went into the destination building. These results are slightly different from those in case 3 of the first experiment. I surmise that causes, such as the geometry of the building, time spent in each environment and the speed at which the user was walking, played a role in the time difference. It was also observed that PENDING was able to handle updating the cyber-model look-up table 100% of the time whenever a new destination was discovered. While detecting user lateness to unknown locations wasn't one of our challenges, we added a feature to PENDING that allows it to dynamically update the cyber table that stores the hash pairs of destination names and GPS coordinates.

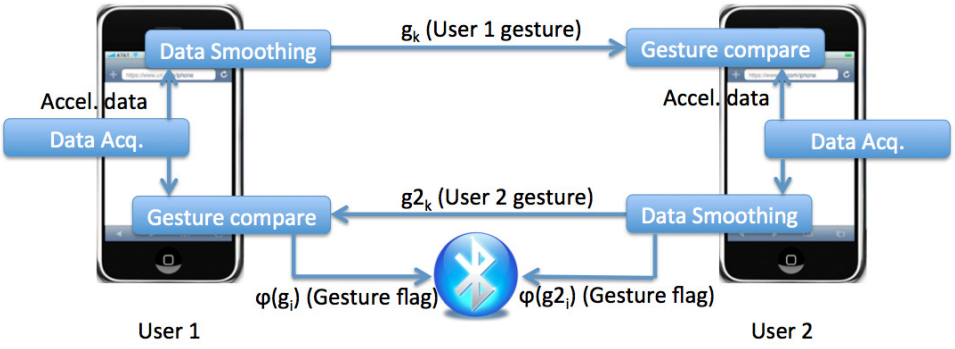
Chapter 6

Authentication Protocol Overviews

6.1 Gesture-based End-User Authentication on Smartphones

To provide a solution to the problem domain of gesture-based end-user authentication for P2P Bluetooth communication on smartphones, I implemented four different protocols: (1) jerk differentiation; (2) software debouncing; (3) window and omit, (4) and gesture template thresholding. Each end-user authentication protocol uses accelerometer-based gestures to generate a unique identity to be shared with another user for P2P Bluetooth communication. Each of these protocols address the three challenges described in Chapter 3: (1) balancing usability and security; (2) making use of jittery smartphone sensor data; and (3) delineating between trusted and untrusted users. I provide high-level descriptions of algorithms and features, formal models, test plans, results and security evaluations for each protocol.

Figure 6.1: A high-level diagram of the Bluehand protocol.



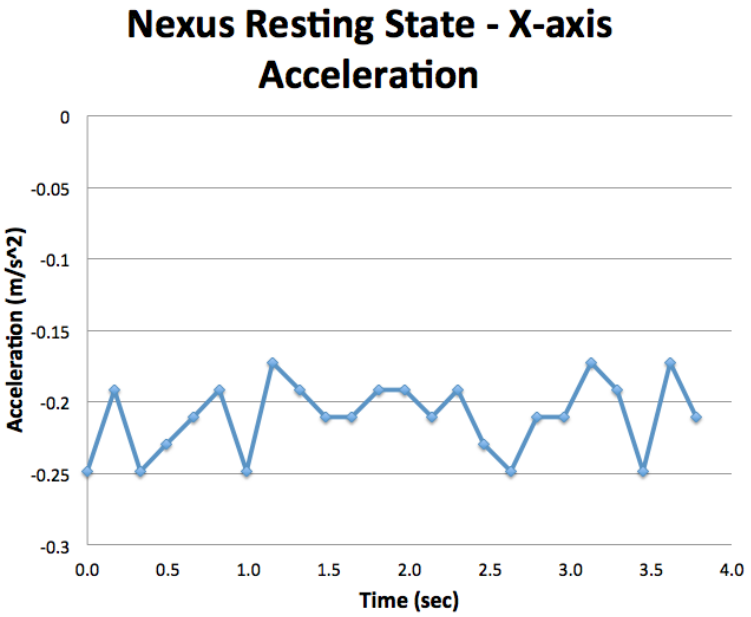
6.2 Gesture Template Matching

The first end-user authentication protocol is based upon the process of gesture template matching, called Bluehand. The premise behind the protocol is that two users who wish to share data with each other over Bluetooth generate a shared key by waving their phone in one of sixteen different gestures; eight gestures and their more exaggerated equivalents as seen in Figure 7.3. If the gesture matches the template of the gesture that the user selected from the set-up screen, seen in Figure 6.1, then the user can pair with another smartphone over Bluetooth. Once paired, the two users share several pieces of information: (1) the gesture that each user selected; (2) processed acceleration data generated by the gesture; and (3) the security threshold that each user selected. Further discussion on each of these pieces of information are below.

6.2.1 Bluehand Data Smoothing

As described in Section 2.4.1, balancing usability and security is a major challenge for end-user authentication on smartphones. The process that the user follows to establish a unique identity on a smartphone must be intuitive and easy to use. The process must also provide

Figure 6.2: Observed smartphone accelerometer jitter when phone is at rest.



a level of security that will prevent end-user impersonation attacks, but not to the point where it becomes impossible for a user to replicate a gesture.

Bluehand's data preprocessing consists of two parts, gesture characterization and gesture comparison. Gesture characterization addresses the usability and security challenge and partially addresses the jittery smartphone sensor data challenge. Gesture comparison addresses the jittery data challenge described in Section 2.4.2. I observed that the raw data of the accelerometer was jittery so I needed to smoothen out the data before comparing two simultaneous gestures. In fact, as seen as an example in Figure 6.2, a smartphone at rest sitting on a table exhibits jitter when an ideal accelerometer would read zero acceleration. By processing the acceleration data to account for jitter, a more accurate data set was acheived.

6.2.2 Bluehand's Formal Model

End-user authentication with gestures requires the processing of physical stimulus data from a smartphone's sensors and comparison of simultaneous gestures to establish secure identity. Bluehand's authentication algorithm is based on a 7-tuple state representation that encodes physical stimulus data and gesture comparison metrics.

$$B_k =< g_k, g2_k, s_k, h_k, T(g_i, g2_i), D(g_i, g2_i), \phi(g_i, g2_i) > \tag{6.1}$$

where:

- $g_k \subset G$ is the current physical gesture the user does to establish secure identity. The set of all gestures is G.
- $g2_k \subset G$ is the current physical gesture the other user does to establish its secure identity. The set of all gestures is G.
- $s_k \subset S$ is the current sensor being used to track the current gesture. The set of all gesture tracking sensors is S.
- $h_k \subset H$ is the current threshold value for gesture comparison. The set of all threshold values is H.
- $T(g_i, g2_i)$ is the expected amount of time to complete the current physical gesture for both users, given $g_i = g2_i$.
- $D(g_i, g2_i)$ is the acceleration data for the gestures captured by s_k.
- $\phi(g_i, g2_i)$ is a flag indicating whether the two gestures were similar enough to authenticate based upon the current threshold value, h_k.

Bluehand is modeled as a function, $Blue(B)$, where the determination of whether or not two users can trust each other in a P2P Bluetooth communication scheme is based on the current gestures provided by the users and the threshold value agreed upon by the users:

$$Blue(B_k) = \begin{cases} T(g_i) - T(g2_i) = 0, g_i = g2_i & 1\ (a), \\ \Delta(D(g_i) - D(g2_i)) < h_k & 1\ (b), \\ \Delta(D(g_i) - D(g2_i)) > h_k & 0\ (c), \end{cases} \tag{6.2}$$

where:

- B_k is a current Bluehand authentication session.
- $\Delta(D(g_i) - D(g2_i))$ is the calculated difference between the sensor data for each user's gesture.

6.2.3 Bluehand Gesture Comparison

The Bluehand end-user authentication protocol considers two users' gestures to be similar enough in two primary cases. In case (a), the amount of time that it takes for each user to complete the gesture is exactly the same. Additionally, the physical gestures are exactly the same. Case (a) is theoretical and never likely to occur; representing an ideal Bluehand session. Case (b) describes Bluehand authentication sessions where the timing and sensor data for each simultaenous gestures are not the same, but the difference is within an agreed upon threshold value.

Figure 6.3: The title screen of the Bluehand Android application.

6.2.4 Bluehand User Control

The second major component of our solution approach is the range of options we gave the user in selecting a pairing gesture and security threshold level. Figure 6.3 is a screenshot of the Bluehand Android application we developed. The users can select one of sixteen gestures and one of three security thresholds. Even though end-user authentication protocols normally fall within a dichotomy of two states, either secure or non-secure, the options were included to evaluate the gesture recognition success rates for various threshold levels and examine the security offerings of an additional parameter. For two users to be able to pair up, the two parameters must be the same.

6.3 Jerk Differentiation

The second end-user authentication protocol that was implemented is based upon jerk differentiation, called Jerky. Points of jerk over time occur when the observed rate of acceleration in a certain direction changes from positive to negative. In this protocol, the algorithm detects all points of jerk during a user's gesture and records the time differences between adjacent jerk points and uses these values for the basis of gesture comparison between two smartphones. Jerk is mathematically defined as:

$$j = \frac{da}{dt} \tag{6.3}$$

which is the differentiation of acceleration in respect to time. By comparing the slope of the lines that connect point x to point x - δt and point x to point x + δt, the algorithm determines whether or not point x is a possible jerk candidate.

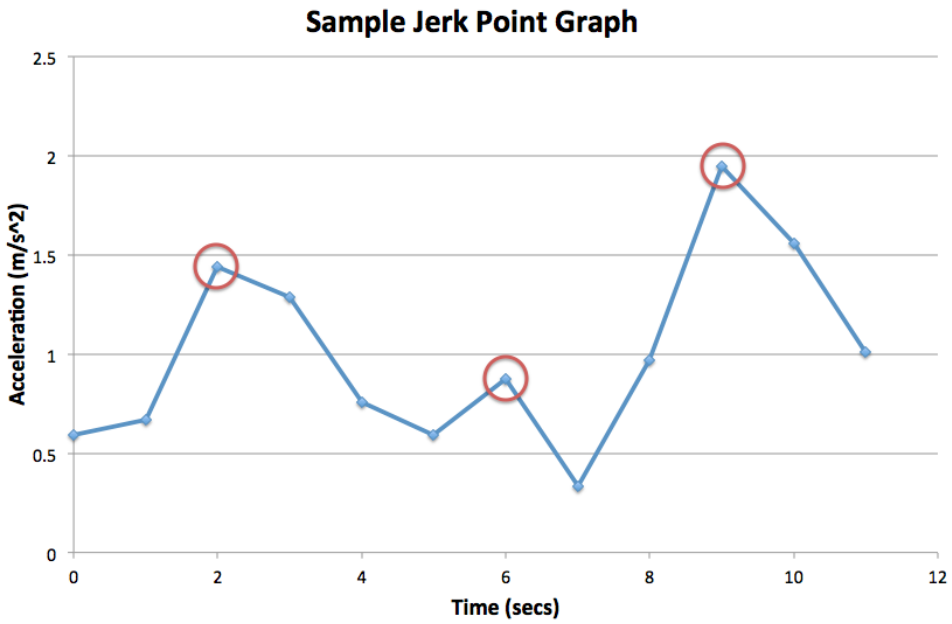
Rather than detecting points of jerk in all three coordinate axes, Jerky creates a vector from the three acceleration datastreams by:

$$A = \sqrt{x^2 + y^2 + z^2} \tag{6.4}$$

The vector all three acceleration values simplifies the algorithm and still maintains accurate jerk detection. Using the new acceleration representation, the two slope equations used to find jerk points is:

$$\frac{A_2 - A_1}{t_2 - t_1}, \frac{A_3 - A_2}{t_3 - t_2} \tag{6.5}$$

Figure 6.4: Example graph of identifiable points of jerk.



where A_1, A_2, and A_3 are three consecutive acceleration vector points and t_1, t_2, and t_3 are their respective timestamps. The algorithm considers the point A_2 at timestamp t_2 as a possible point of jerk when:

$$\frac{A_2 - A_1}{t_2 - t_1} > 0 \;\&\&\; \frac{A_3 - A_2}{t_3 - t_2} < 0 \tag{6.6}$$

where the the slope leading into A_2 is rising and the slope leaving A_2 is descending. Figure 6.4 is an example graph showing possible jerk points at t=2, 6, and 9 seconds.

Though points that meet the above conditions are mathematically considered as jerks, not all jerk points are interesting or useful. Recall that the accelerometer in smartphones exhibit jitter that would generate points of jerk even though the device is not physically moving. To account for this possibility, a minimum acceleration threshold must be met before a point can be considered as a jerk point candidate.

6.4 Jerky's Formal Model

Jerky requires the processing of physical stimulus data from a smartphone's accelerometer and comparison of simultaneous gestures to establish secure identity. Jerky's authenticaiton algorithm is based on a 9-tuple state representation that encodes physical stimulus data and gesture comparison metrics.

$$J_k =< g_k, g2_k, n_k, h_k, t_k, A(g_i, g2_i), D(g_i, g2_i), T(g_i, g2_i), \phi(g_i, g2_i) > \tag{6.7}$$

where:

- $g_k \subset G$ is the current physical gesture the user does to establish secure identity. The set of all gestures is G.
- $g2_k \subset G$ is the current physical gesture the other user does to establish its secure identity. The set of all gestures is G.
- $n_k \subset N$ is the number of observed jerks necessary to establish a unique identity. The set of all possible numbers is S.
- $h_k \subset H$ is the minimum acceleration threshold for a jerk point to be considered useful. The set of all possible thresholds is H.
- $t_k \subset T$ is the maximum time difference threshold for comparing relative time stamps.
- $A(g_i, g2_i)$ is the acceleration data for the gestures captured by s_k.
- $D(g_i, g2_i)$ is the jerk data for the gestures by differentiating $A(g_i, g2_i)$.
- $T(g_i, g2_i)$ is the time differences between adjacent jerk points.

- $\phi(g_i, g2_i)$ is a flag indicating whether the two gestures generated the same number of jerks as described by n_k and obtained by $D(g_i, g2_i)$, and whose relative time differences fall within t_k .

Jerky is modeled as a function, Jerk(S), where the determination of whether or not two users can trust each other in a P2P Bluetooth communication scheme is based on the current gestures provided by the users, the number of jerk points needed to authenticate and the timing characteristics of the two users' gestures:

$$Jerk(J_k) = \begin{cases} D(g_i) = n_k \;\&\&\; D(g2_i) = n_k \;\&\& \\ A(g_i, t_i) > h_k \;\&\&\; A(g2_i, t2_i) > h_k \;\&\&\; T(g_i, g2_i) < t_k & 1\ (a), \\ D(g_i) \neq n_k \;\|\; D(g2_i) \neq n_k \;\|\; A(g_i, t_i) < h_k \;\| \\ A(g2_i, t2_i) < h_k \;\|\; T(g_i, g2_i) > t_k & 0\ (b), \end{cases} \tag{6.8}$$

where:

- J_k is a current Jerky authentication session.
- $D(g_i) = n_k$ is the expansion of $\phi(g_i)$, the flag indicating if two gestures have the same number of jerks and similar enough timing characteristics.
- $A(g_i, t_i) > h_k$ states that an acceleration point must be above the minimum threshold to be considered as a jerk point.
- $T(g_i, g2_i) < t_k$ states that the relative time differences between two sets of adjacent jerk points must be less than

6.4.1 Jerky Gesture Comparison

The Jerky end-user authentication protocol considers two users' gestures to be similar enough in one case. In case (a), the number of observed jerks in the acceleration data for both gestures is the same, as set by n_k, the jerk point has a corresponding acceleration greater than h_k, and the relative time differences between the two gestures are less than t_k. Case (b) describes Jerky authentication sessions where the number of observed jerks for the two gestures are not the same, or the jerk point does not have a corresponding acceleration value greater than h_k, or the relative time difference between the two gestures is greater than t_k.

6.5 Software Debouncing

Due to the jittery nature of hardware buttons, a technique known as debouncing is applied to signals to smoothen out the transitions between high and low logic states to pinpoint intentional button presses outside of mechanical error. The third end-user authentication protocol, called Smoother, is based upon the same principal to mitigate the jittery nature of the smartphone accelerometer. In this protocol, the algorithm detects "hills" in the accelerometer data stream to find local maxima peak points for comparison against the peak points of the other user's gesture. Consider five sequential acceleration values provided by an accelerometer:

$$a_1 = 3.41, a_2 = 4.89, a_3 = 9.03, a_4 = 5.21, a_5 = 2.89 \tag{6.9}$$

The data stream represents a hill, where point a_3 is the peak point since a_1 less than a_2, a_2 less than a_3, a_3 greater than a_4, and a_4 greater than a_5. By comparing the values of previous and subsequent points to the local maxima, the data can be smoothed to compensate for

accidental hills in jittery accelerometer data. The process is generalized as follows:

$$a_k < a_k + 1, a_m > a_m + 1, k < m < n \tag{6.10}$$

Similar to Jerky, Smoother creates a vector from the three accelerometer datastreams described in Equation 6.4. The new acceleration vector values are shifted one at a time through the debouncer to detect any peaks that can be used for gesture comparison. Further discussion on how the size of the debouncer was selected is described later.

6.5.1 Smoother Formal Model

Smoother's authenticaiton algorithm is based on a 8-tuple state representation that encodes physical stimulus data and gesture comparison metrics.

$$S_k =< g_k, g2_k, s_k, n_k, h_k, A(g_i, g2_i), P(g_i, g2_i), \phi(g_i, g2_i) > \tag{6.11}$$

where:

- $g_k \subset G$ is the current physical gesture the user does to establish secure identity. The set of all gestures is G.
- $g2_k \subset G$ is the current physical gesture the other user does to establish its secure identity. The set of all gestures is G.
- $s_k \subset S$ is the size of the software debouncer. The set of all possible sizes is S.
- $n_k \subset N$ is the number of peak points necessary to establish identity. The set of all numbers is N.

- $h_k \subset H$ is the minimum acceleration threshold value for a point to be considered a peak point.
- $A(g_i, g2_i)$ is the acceleration data for the gestures.
- $P(g_i, g2_i)$ is the smoothing algorithm on $A(g_i, g2_i)$ to find all possible peak values.
- $\phi(g_i, g2_i)$ is a flag indicating whether the two gestures generated similar peak points.

Smoother is modeled as a function, Smooth(S), where the determination of whether or not two users can trust each other in a P2P Bluetooth communication scheme is based on the current gestures provided by the users and number of peak points needed to authenticate:

$$Smooth(S_K) = \begin{cases} P(g_i) = n_k \;\&\&\; P(g2_1) = n_k \;\&\&\; A(g_i, t_i) > h_k \;\&\&\; A(g2_i, t2_i) > h_k & 1\ (a) \\ P(g_i) \neq n_k \;\parallel\; P(g2_1) \neq n_k \;\parallel\; A(g_i, t_i) < h_k \;\parallel\; A(g2_i, t2_i) < h_k & 1(b) \end{cases} \tag{6.12}$$

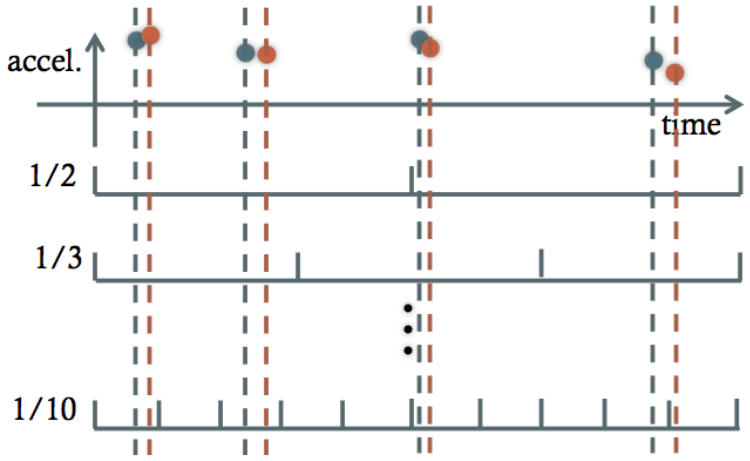
where:

- S_k is a current Smoother authentication session.
- $P(g_i) = n_k$ is the expansion of $\phi(g_i)$, the flag indicating if two gestures have the same number of peak points.

6.5.2 Smoother Gesture Comparison

The Smoother end-user authentication protocol considers two users' gestures to be similar enough in one case. In case (a), the number of observed peak points in the acceleration data

Figure 6.5: The process of relative windowing.



for both gestures are the same, as set by n_k. Case (b) describes Smoother authentication sessions where the number of observed peak points for the two gestures are not the same.

6.6 Window and Omit

The fourth end-user authentication protocol is an extention Jerky protocol. Recall the in the Jerky protocol, the time difference pairs of adjacent jerk points were compared against t_k to determine if two gestures are considered similar enough (i.e. $\phi(g_i, g2_i) = 1$). Rather than using the timestamps of the jerk points in the same manner, the fourth protocol, called WindowAndOmit, splices the data set into subsets by using relative windowing. The idea behind relative windowing is illustrated in Figure 6.5.

WindowAndOmit creates nine subsets of the original data set by specifying relative windows of time where pairs of jerk points provided by the two smartphones are compared within their respective time windows. If a jerk point of one phone is not accompanied by a jerk point of the other phone within a certain time window, then the entire subset is omitted. For two gestures to be considered similar enough, none of the subsets can be omitted.

6.7 WindowAndOmit Formal Model

WindowAndOmit's authenticaiton algorithm is based on a 9-tuple state representation that encodes physical stimulus data and gesture comparison metrics.

$$W_k =< g_k, g2_k, s_k, r_k, n_k, h_k, A(g_i, g2_i), P(g_i, g2_i), \phi(g_i, g2_i) > \tag{6.13}$$

where:

- $g_k \subset G$ is the current physical gesture the user does to establish secure identity. The set of all gestures is G.
- $g2_k \subset G$ is the current physical gesture the other user does to establish its secure identity. The set of all gestures is G.
- $s_k \subset S$ is the number of windowed subsets. The set of all possible numbers is S.
- $r_k \subset R$ is the relative window width of the current subset. The set of all possible widths is R.
- $n_k \subset N$ is the number of peak points necessary to establish identity. The set of all numbers is N.
- $h_k \subset H$ is the minimum acceleration threshold value for a point to be considered a peak point.
- $A(g_i, g2_i)$ is the acceleration data for the gestures.
- $P(g_i, g2_i)$ is the smoothing algorithm on $A(g_i, g2_i)$ to find all possible peak values.
- $\phi(g_i, g2_i)$ is a flag indicating whether the two gestures generated similar peak points.

Chapter 7

Bluehand Test Plan and Results

7.1 Experimental Platform

I ran several tests and experiments to evaluate Bluehand's ability to recognize gestures and discriminate between trusted and untrusted users. I first compared the speed of entering a randomly-generated 4-digit Bluetootn PIN on a smartphone touch screen versus the speed of generating a 4-digit PIN by conducting a Bluehand gesture. I then characterized the gestures and security thresholds to be used through statistical extraction and evaluation. I then tested Bluehand's success rate at recognizing trusted and untrusted users through a series of pairing sessions. All tests were done on Android smartphones installed with the Bluehand application.

7.2 Speed Comparisons

I wrote an Android application that measured the time it took a user to manually input a randomly-generated 4-digit PIN on a smartphone touchscreen. I ran twenty-five trials to

Figure 7.1: Results from the manual PIN entry speed test.

Minimum time:	0.872 sec
Maximum time:	2.357 sec
Average time:	1.504 sec
Standard deviation time:	0.349 sec

obtain a minimum time, maximum time, average time, and standard deviation time of the data set, which can be seen in Figure 7.1.

I wrote another Android application that measured the time it took a user to conduct one of the sixteen gestures for Bluehand to generate a 4-digit PIN. I ran five trails of each of the sixteen gestures to obtain the minimum time, maximum time, average time and standard deviation time for each gesture, as seen in Figure 7.2. I divided the time it took to complete the gesture into four time slices, where each time slice was used as a seed to generate the corresponding digit in a 4-digit PIN.

Figure 7.2: Results from the gesture-based PIN generation speed test.

	Min. time (sec)	Max. time	Average time	Standard deviation time
small right	1.148	1.719	1.330	0.228
small left	0.991	1.532	1.210	0.209
small up	0.904	1.355	1.158	0.173
small circle	1.234	1.995	1.641	0.301
small plus	1.144	1.809	1.433	0.247
small zigzag	1.323	1.462	1.394	0.054
small u	1.168	1.413	1.323	0.104
small diagonal	0.937	1.132	1.006	0.076
big right	0.797	0.936	0.840	0.058
big left	0.797	1.130	0.942	0.125
big up	0.710	0.937	0.828	0.083
big circle	0.932	1.615	1.307	0.264
big plus	1.164	1.550	1.375	0.169
big zigzag	1.074	1.601	1.379	0.219
big u	1.108	1.424	1.228	0.130
big diagonal	0.727	1.025	0.915	0.119

The small eight gestures required slightly more time to complete than their larger counterparts. The large gestures were more exagerrated versions where the user may not be as concerned about conducting the gesture precisely. The average time values to complete the larger gestures were all smaller than those of the small gestures.

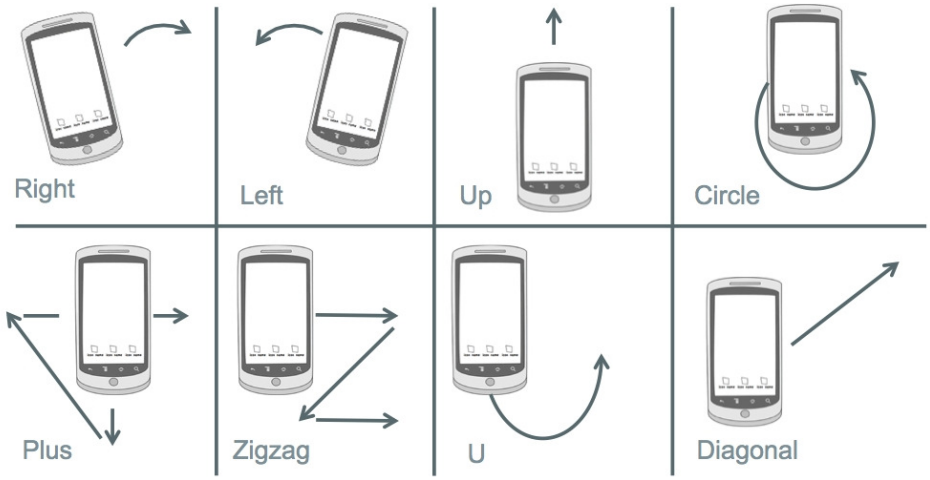
I see that the average time for the user to manual input a 4-digit PIN on a smartphone touch screen is 1.504 seconds. The user typed the number in with the thumb while holding the smartphone device with the dominant writing hand. Fifteen of the sixteen gestures had lower average time values than the average time to manually enter a 4-digit PIN. This means that it is generally quicker for a user to wave the smartphone to generate a 4-digit PIN rather than to manually input one via touchscreen.

7.3 Gesture Characterization

Before we tested Bluehand's functionality, we characterized sixteen gestures that the user could choose from during a Bluehand end-user authentication session. The gestures included a right flick motion, left flick motion, upward motion, circle, plus-sign motion, zigzag motion, U-shaped motion, an upward diagonal motion and larger, exaggerated versions of these eight. The motion sequences can be seen in Figure 7.3. I chose these gestures because of their relative uniqueness from one another and for the shrot amount of time needed to conduct the gesture.

It is important to note that some of the gestures move in all three cartesian directions while others only move in two. The difference in motion vectors does play an important role in gesture recongition, which we discuss later on. To gain a better understanding of the physical qualities of each of these gestures, we wrote a data logging Android application that monitored the average x-, y-, z-acceleration. The application would start recording the

Figure 7.3: Motion maps for Bluehands eight small gestures.



accelerometer data when the user pressed a start button and would stop recording when the user pressed the stop button.

I ran ten trials of each gesture and at the end of each trial, the application averaged the recorded accelerometer data to aqcuire an average acceleration value for each axis. The values acquired from this test are summarized in Figure 7.4.

7.4 Threshold Benchmarking

Using the average acceleration values from our gesture characterization test, we ran another test to find threshold candidates for gesture comparison during a Bluehand session. I evaluated six different threshold levels that were multiplicative factors of the standard deviations of the x-, y- and z-acceleration for each gesture. To evaluate the threshold candidates, we wrote a second Android application that determined if a particular gesture fell within an acceptable range based upon the current threshold level.

The code for the application was based upon a set of boundary-defining equations that

Figure 7.4: Average values used to characterize Bluehand's eight small gestures.

	Avg(X)	Avg(Y)	Avg(Z)
Right	0.15348	1.64078	8.93194
Left	-0.09537	1.62481	8.95998
Up	-0.24484	9.87822	0.68596
Circle	-0.12065	9.64925	1.57706
Plus	-0.17395	9.84524	1.02921
Zigzag	-0.43712	9.61466	1.91792
U	-0.157607	9.68651	1.62626
Diagonal	-0.4657	9.74339	1.07615

described the upper and lower limit for the average x-, y- and z-acceleration of a particular gesture. If any of the observed values fell outside these ranges, then the gesture was not recognized. Smaller thresholds made it more difficult for a gesture to be recognized, while larger thresholds made it easier for a gesture to be recognized. Below are the boundary-defining equations:

$$T(g_i) = \begin{cases} a_x(g_i) < a_x(g) + t_x(g)\&\& \\ a_x(g_i) > a_x(g) - t_x(g)\&\& \\ a_y(g_i) < a_y(g) + t_y(g)\&\& \\ a_y(g_i) > a_y(g) - t_y(g)\&\& \\ a_z(g_i) < a_z(g) + t_z(g)\&\& \\ a_z(g_i) > a_z(g) - t_z(g) & 1\ (a) \\ otherwise & 0\ (b) \end{cases} \tag{7.1}$$

The boundary equation, $T(g_i)$, determines if the current gesture, g_i, is a recognized gesture

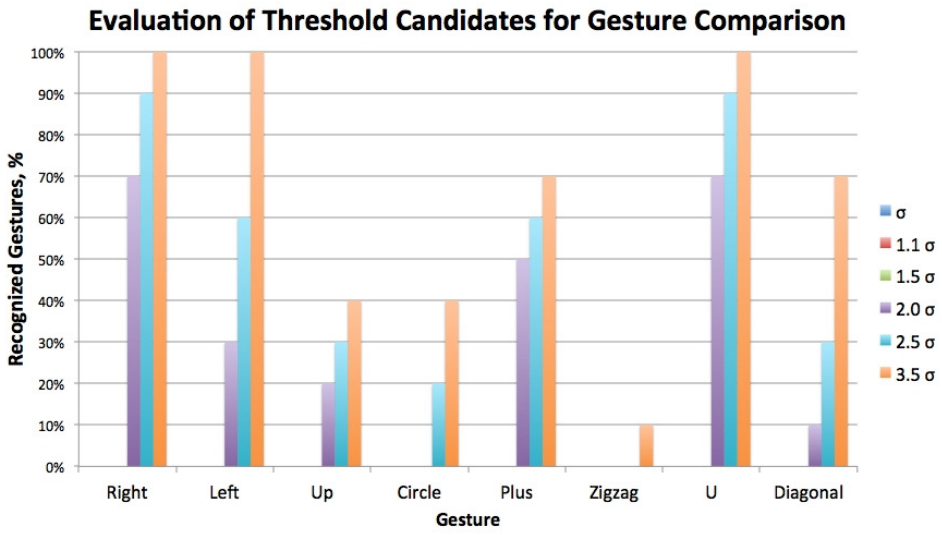
according to the template gesture, g. The following terms are elements in the boundary equation:

- $a_x(g_i)$ is the average x-acceleration of the current gesture, g_i.
- $a_x(g)$ is the average x-acceleration of the template gesture, g.
- $t_x(g)$ is the current x-acceleration threshold of the template gesture, g.
- $a_y(g_i)$ is the average y-acceleration of the current gesture, g_i.
- $a_y(g)$ is the average y-acceleration of the template gesture, g.
- $t_y(g)$ is the current y-acceleration threshold of the template gesture, g.
- $a_z(g_i)$ is the average z-acceleration of the current gesture, g_i.
- $a_z(g)$ is the average z-acceleration of the template gesture, g.
- $t_z(g)$ is the current z-accerlation threshold of the template gesture, g.

Figure 7.5 shows our evaluation of threshold candidates based upon the standard deviation of the accelerometer data. I observed that threshold values σ, 1.1 * σ and 1.5 * σ were too stringent for any gesture to be recognized, where σ is the standard deviation of the acceleration. I saw an increase in gesture recongition as we increased the threshold values up to 2.0 * σ, 2.5 * σ and 3.5 * σ. The three latter threshold values correspond to three levels of security that we discuss later.

I also observed that certain gestures were more consistently recognized than others, such as the right-flick, left-flick, and U-shaped gesture. The orange bar represents the gesture recognition rate at the highest threshold value.

Figure 7.5: Evaluation of possible threshold candidates.



I mapped the three threshold values that yielded gesture recognition to relative security levels. Figure 7.6 shows the threshold level mapping. The threshold values are used to compare the difference between the acceleration values of two simultaneous gestures to determine if they are similar enough to each other. At the beginning of a Bluehand session, the users will negotiate upon a security level before proceeding to gesture comparison.

Figure 7.6: Security level mapping to threshold values.

	Accelerometer
Top Secret (TS)	2.0 * σ
Secret (S)	2.5 * σ
Unclassified (U)	3.5 * σ

7.5 Bluehand Experiments

Figure 6.3 is a screenshot of Bluehand's title screen where the user to selects a gesture and a security level. Once the parameters are set, both users can enter a Bluehand session where

they first pair up over Bluetooth. Before any file or textual data can be sent, each user must complete their gesture and then send the data for comparison to each other. If the gestures are similar enough based upon the agreed security level, then the users will be able to send data to each other. If the gestures are not similar enough, then the two users are unpaired and the application reverts the users back to the beginning of the application.

I conducted two experiments to test Bluehand's functionality. The metrics of evaluation were Bluehand's gesture recognition success rate and Bluehand's false positive rate.

7.5.1 Verifying trusted users experiment

The first experiment was designed to test Bluehand's gesture recognition. Each user selected a gesture and security level to serve as the threshold for comparison. Once each phone was paired up over Bluetooth, the users move their phones and then send the data to each other. The gesture comparison algorithm measures the difference between the gesture data and if the difference falls within the threshold level then the users are provided an open channel of P2P Bluetooth communciation.

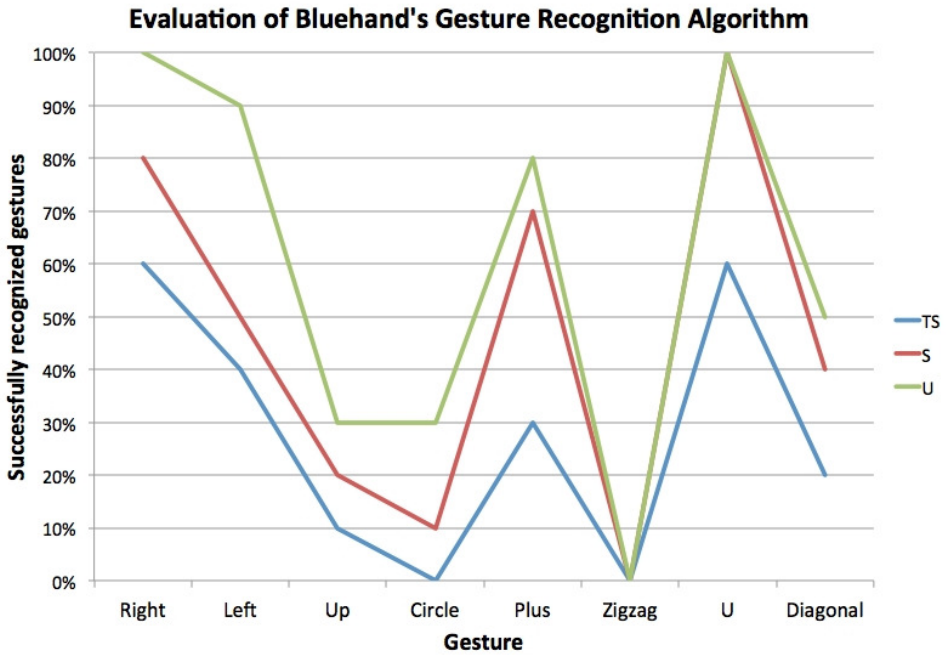
I ran ten trials of each of the eight small gestures for each of the three threshold levels, resulting in a sample size of n = 240. The success rates for each trial, which correlates to the probability that Bluehand could recognize a trusted user, were recorded for later analysis.

Hypothesis: There is a correlation between thresholds and gesture matching success rate. My hypothesis was that Bluehand's gesture recognition would follow the same trend of our threshold benchmarking test. I expected that the users would be able to authenticate more often when the threshold level was larger, and conversely for smaller threshold levels.

Verifying trusted users experiment results. Figure 7.7 shows how Bluehand's gesture

recognition changes from one gesture and security level to another. Recall from Figure 7.6 that the three possible security levels are top secret (TS), secret (T) and unclassified (U), corresponding to threshold levels 2.0 * σ, 2.5 * σ and 3.5 * σ, respectively. When examining the data we realized two important facts. The first fact was that certain gestures were more easily recognized than others. For example, the right, plus and U-shaped gestures were the most recognized gestures across all three security levels. In fact, the right and U-shaped gestures were recognized 100

Figure 7.7: Results of Bluehand's gesture recognition experiment.



The second important fact concerns the increase in gesture recognition from one security level to the next. Even though TS, S, and U are scalar factors of each gestures' acceleration standard deviation, we did not observe the same scalar increases in gesture recognition. For example, gesture recognition for the right gesture follows a general linear trend, but gesture recognition for the plus-sign gesture remained at zero despite the threshold increase. The motion complexity of the gesture plays a role in how easily it is recognized. The plus-sign

gesture exhibits multiple motion segments, so the compounding variability in acceleration for each segment makes it more difficult to recognize as a whole.

7.5.2 Detecting malicious users experiment

The second experiment tested Bluehand's ability to detect malicious users who masquerade as a trusted users. I tested each gesture against the other seven gestures ten times at each of the three threshold levels, which resulted in a sample of n=840 false postive tests. After the users share gesture data, Bluehand's gesture comparing algorithm verifies whether or not the gestures match. If not, the P2P Bluetooth channel is closed and the smartphones are unpaired.

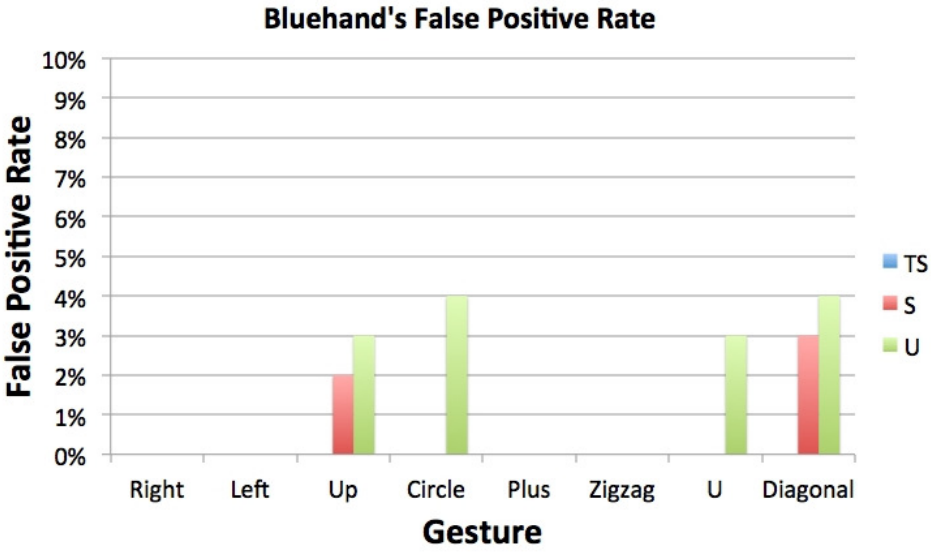
The experiment metric was Bluehand's false positive rate which translates to a malicious user being able to authenticate itself as a trusted user even though the two users' gestures were different. The second experiment directly addresses the challenge of discriminating between trusted and untrusted users described in Section 2.4.3.

Hypothesis: Bluehand detects end-user masquerading. The hypothesis was that Bluehand would be able to effectively discriminate between two different gestures. I expected that when the two users share different enough gesture data, Bluehand would recognize the difference and bar any further communication.

Experiment 2 Results.

Figure 7.8 shows the false positive rate for each gesture at each security level. Four gestures generated false positives at the secret and unclassified security levels: up, circle, U-shaped, and diagonal. Specifically, whenever circle and U-shaped gestures were done simultaneously, Bluehand at times interpreted them as similar enough in regards to the current threshold. The same is true of the up and diagonal gesture. I speculate the cause of the false positives is

Figure 7.8: Results showing Bluehand's false positive rate.



because the gesture pairs exhibit movements in the same cartesian directions and orientation.

The highest false positive rate was 4% of a sample size of n=70 Bluehand sessions at a certain security threshold level. False positives were non-existant for the right, left, plus sign and zigzag gestures at all three security levels and for all gestures at the top secret security level.

Chapter 8

Smoother Test Plan and Results

8.1 Determining Debouncer Size

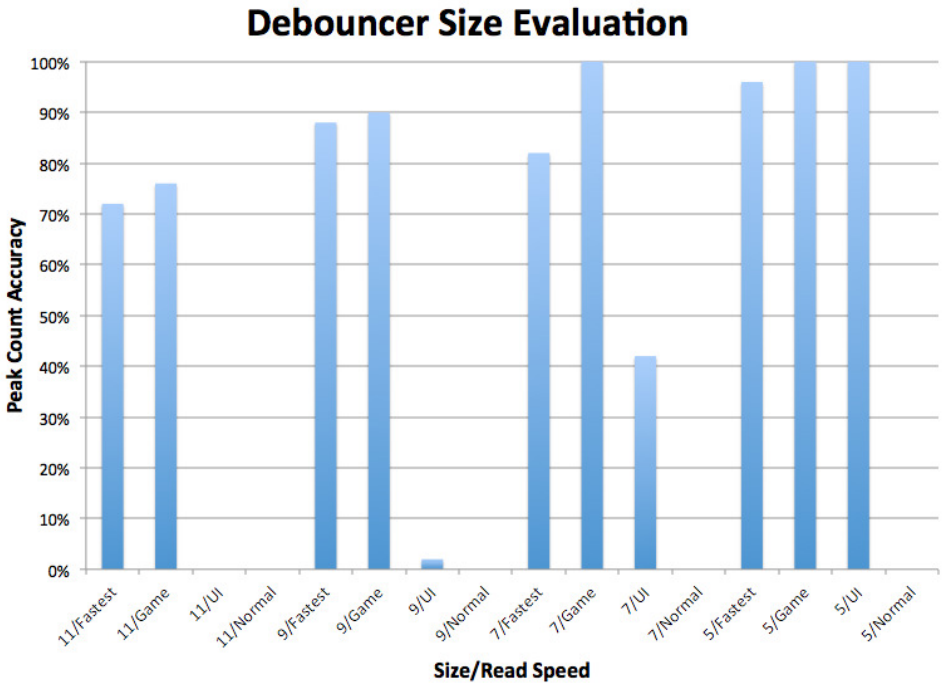
The first step in translating Smoother's formal model into a smartphone application was to determine an appropriate size for the debouncer. Recall that the debouncer takes a series of consecutive acceleration data points provided by the smartphone's accelerometer and determines points of clear directional change. As new acceleration values are available, they are shifted into the debouncer for evaluation. The acceleration values are first vectorized:

```
accelVectorFloat_= FloatMath.sqrt(arg0.values[0]^2 +
                   arg0.values[1]^2 +
                   arg0.values[2]^2);
```

where arg0 is the new set of accelerometer values provided by the smartphone's accelerometer. The new values are shifted in as such:
Whenever a new acceleration point is shifted in, the debouncer checks if there is a clear peak point by verifying that the values preceding the midpoint of the dataset are increasing while the values following the midpoint are decreasing. Four s_k values were tested as candidates for the size of the debouncer, specifically 5, 7, 9 and 11. These sizes translate to the number

```
for(int j = 0; j < accelVectorFloatArray_.length; j++){
    if(j < accelVectorFloatArray_.length - 1){
        accelVectorFloatArray_[j] = accelVectorFloatArray_[j+1];
    }
    else{
        accelVectorFloatArray_[j] = accelVectorFloat_;
    }
}
```

Figure 8.1: Evalution of size candidates for Smoother's debouncer.



of points the debouncer simultaneously smoothens out to find peak points. The minimum acceleration threshold,h_k, was empirically set to 16.00 m/s^2.

I wrote an Android application to test the different combination of debouncer sizes and accelerometer read speeds to determine the most optimal parameter settings. The evaluation of each of these parameter combinations can been seen in Figure 8.1.

In general, as the size of the debouncer increased, the peak detection rate decreased. The slowest of the four Android accelerometer read speeds, SENSOR_DELAY_NORMAL, was not

fast enough to populate the debouncer quickly enough to detect all of the intentional peak points. The SENSOR_DELAY_UI read speed, which is intended for Android applications for user interfacing purposes, was able to detect the peak points 100% of the 50 gesture test run. The SENSOR_DELAY_GAME read speed also yielded 100% peak point detection when the size of the debouncer was at the lower end of the range of size candidates. The SENSOR_DELAY_FASTEST read speed appeared to be too fast, perhaps due to the fact that data was overwritten too quickly before all the data could be processed. To account for any possible false positives, the SENSOR_DELAY_GAME read speed and debouncer size of 7 were chosen as the parameters to define the debouncer.

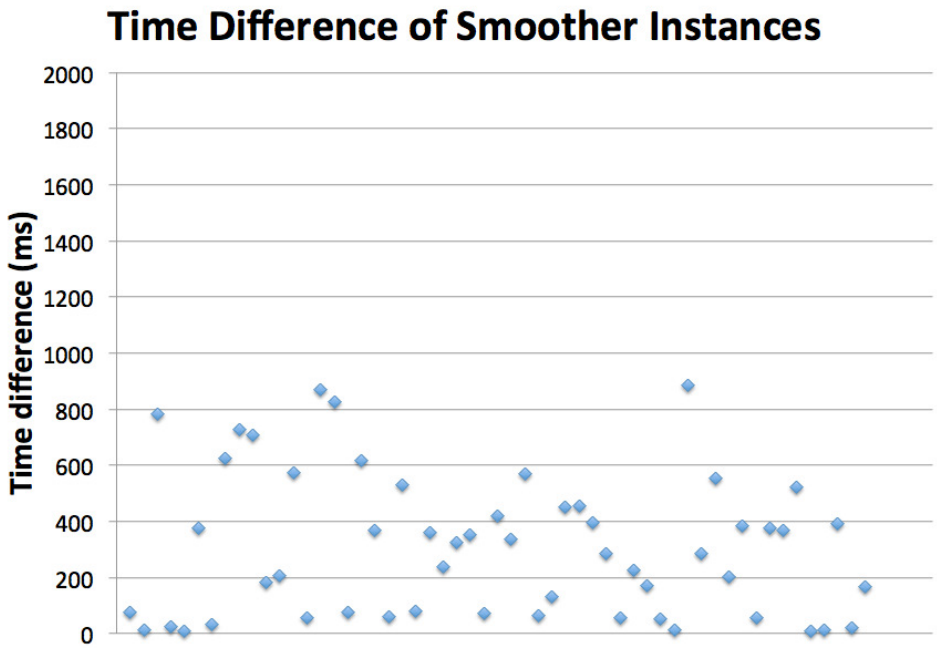
8.2 Smoother Experiments

To test the functionality of Smoother as an end-user authentication protocol for smartphones, I tested how precisely two simultaneous gestures on two different Android smartphones can be matched. Recall from Smoother's Formal Model, that $\phi(g_i, g2_i)$ is the flag that indicates whether or not two simultaneous gestures are similar enough by the equality of the number of peak points. The Smoother application records the timestamp of when the debouncer detects n_k number of peak points. The timestamp is then compared between the two phones to determine if they are close enough to be considered a match.

8.2.1 Smoother Threshold Benchmarking

I used a Droid 3 and a Nexus Android smartphone to test the time difference of when the two simultaneous Smoother instances detected n_k peak points. Fifty trials were conducted where the two phones are held together in one hand and shaken until five peak points are detected. At this point, the elapsed time from the first peak to the fifth peak is recorded, whose values can be seen in Figure 8.2.

Figure 8.2: Time difference between peak point detection by Smoother.

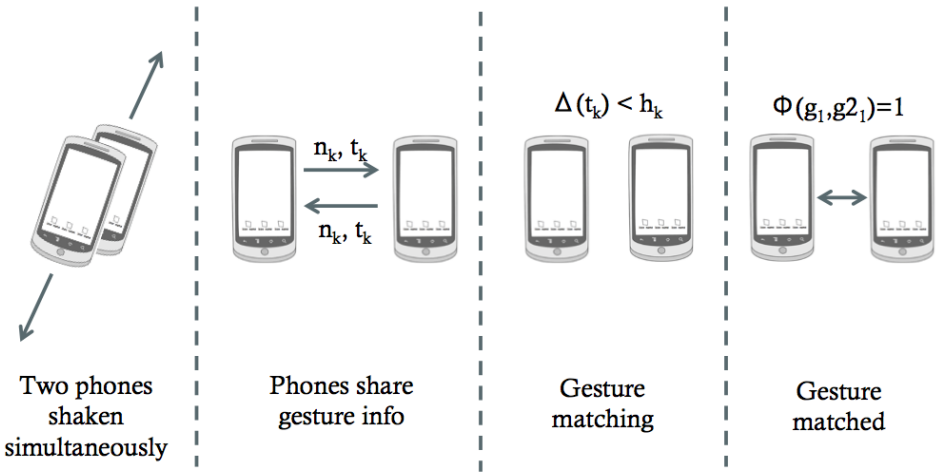


From the data set, the largest difference in detection time was 884 ms, the smallest difference in detection time was 8 ms and the average time difference was 309.65 ms. The average value was then used as the timing threshold metric for determining if two gestures were similar enough, $\phi(g_i, g2_i)$.

8.2.2 Smoother Functionality Tests

Using the same two Android smartphones, tests were conducted to test Smoother's functionality. Figure 8.3 outlines the four stages of the test. The first stage is the two phones are shaken together in one hand for a few seconds. Once each Smoother instance detects n_k peak points, the elapsed time is sent to the other phone. The third stage performs gesture comparison based upon the two t_k values in relation to the timing threshold, h_k. If the gesture parameters meet the conditions as determined by $\phi(g_i, g2_i)$, then the two users can communicate over P2P Bluetooth, otherwise, Bluetooth communication is suspended and

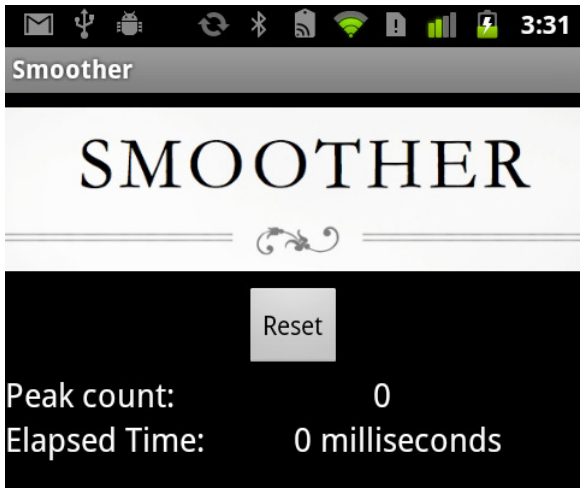
Figure 8.3: Schedule of events for Smoother functionality tests.



application aborts.

I categorized two types of gestures: linear and rotational gestures. The tested linear gestures (+x, -x, +y, -y, +z, -z) exhibit motion in one of three coordinate axes. The rotational gestures (rotate x, rotate y, rotate z) exhibit motion around a coordinate axis. Ten trials of each gesture were tested resulting in ninety Smoother trials. The title screen of the application is seen in Figure 8.4. The accelerometer and Bluetooth adapter are enabled and the background timer initiates when the debouncer detects the first peak point. After n_k peak points are observed, the timer records the elapsed time and a new Android activity is started which transitions to a Bluetooth-based P2P chat application. The two phones connect to each other over Bluetooth, and once the state handler thread detects a connected state, the gesture timing information is shared. Smoother then checks for a gesture match using the empirically determined timing threshold, h_k. If Smoother determines that the gestures are similar enough, then the two users can continue to chat over P2P Bluetooth, otherwise the chat application is terminated and the Bluetooth adapter automatically disabled to prevent any Bluetooth attacks.

Figure 8.4: Smoother Android application title screen.

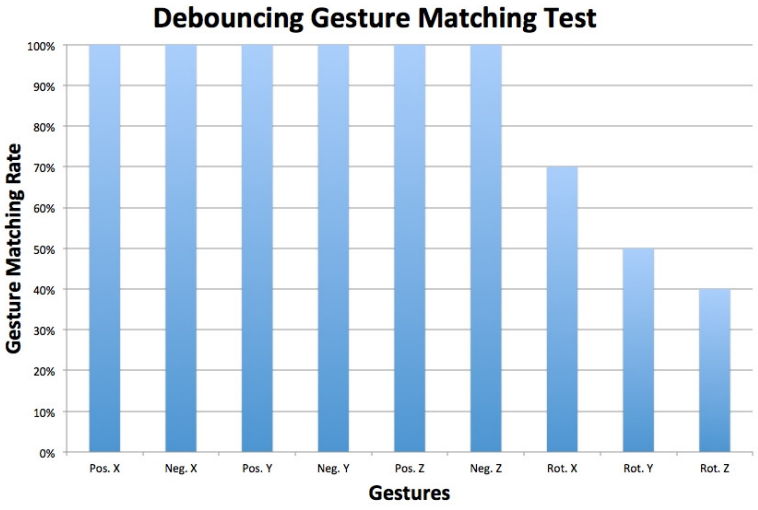


The results of the described tests can be graphically seen in Figure 8.5. The application was able to correctly identify 100% of the linear gesture motions. The rotational gestures resulted in lower gesture matching rates due to the possible difference in accelerometer placement in Droid 3 and Nexus S smartphones. Linear motions would result in near identical acceleration values from two phones moved together in the same hand where rotational gestures can results in accelerometer variability.

The false negative rate is the reciprocal of the gesture matching rate, where the linear gestures resulted in zero fase negatives.

The second experiment evaluated the false positive rate of each of the three protocols. I defined a false positive as a gesture match occurring between two gestures that do not exhibit the same motion but have similar acceleration and timing characteristics. In this experiment, two users each moved their phone with no knowledge of the motion path and timing of the other users gesture. I predicted that the library size of possible gestures would be large enough to result in a near zero false positive rate. After running fifty trials, the false positive rate was 0%. There were no incidental matches between two randomly selected

Figure 8.5: Smoother gesture matching results.



gestures by the two users.

The third test compared the speed entry of manually entered 4-digit PINs versus the average speed of gesture entry for our three end-user authentication protocols. I wrote an Android application that generates a random 4-digit number and measures the total time it takes of the user to type it in on the smartphone touchscreen. An average PIN takes 1.804 +/- 0.301 seconds to input on a smartphone touchscreen, an average gesture with the Smoother protocol takes 1.766 +/- 0.221 seconds. The difference is subtle but the gesture protocols take slightly less time to acquire the necessary amount of acceleration data to generate a PIN.

Chapter 9

Jerky Test Plan and Results

9.1 Jerky Threshold Benchmarking

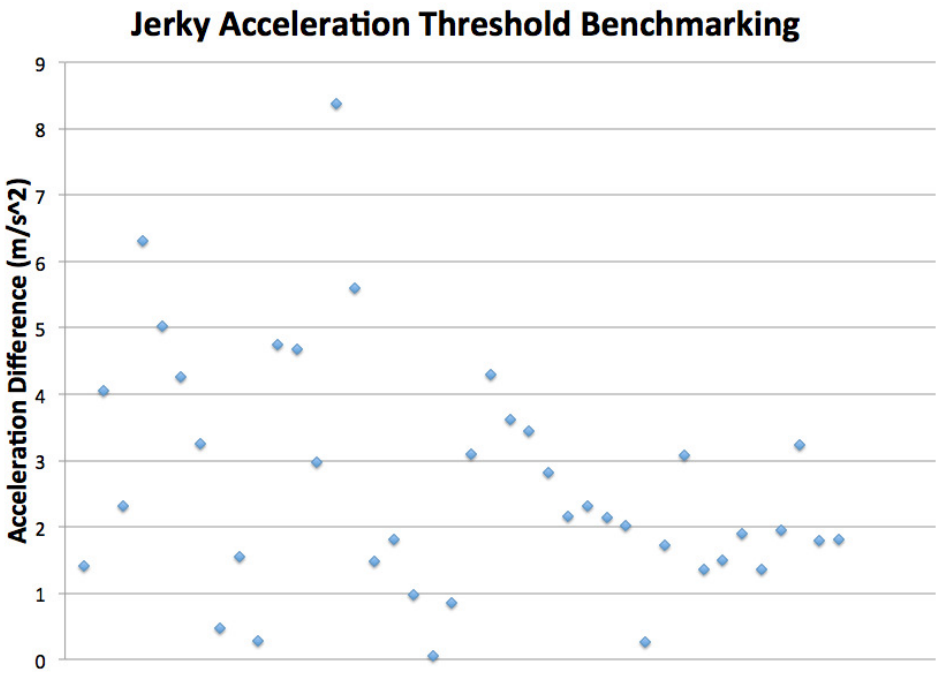
To address the jittery accelerometer data challenge of end-user authentication protocols on smartphones, the jerk point, h_k, and relative time difference, t_k, thresholds were determined by observing initial data sets of accelerometer and timestamps. An Android application built in the same manner as Smoother was installed on the same two smartphones, Droid 3 and Nexus S. After running several trial runs and observing the data, h_k was empirically set to 5.42 m/s^2 and t_k was set to 300 ms. The results of the benchmarking test can be seen in Figure 9.1.

The acceleration difference threshold is mathematically defined as:

$$h_k = \frac{\sum_{i=1}^{n} \Delta a_i}{n} + \sigma(a_0, a_1, ..., a_n) \tag{9.1}$$

where n is the total number of acceleration point pairs and σ is one standard deviation of

Figure 9.1: Jerky acceleration threshold benchmarking result.



the data set. Figure 9.2 shows a visual representation of how the timing threshold, t_k was determined.

The time difference threshold is mathematically defined as:

$$t_k = \frac{\sum i = 1^n (t_{i+1} - t_i) - (t2_{i+1} - t2_i)}{n} + \sigma(\Delta t_0, \Delta t_1, ...\Delta t_n) \tag{9.2}$$

where n is the total number of adjacent time difference pairs and σ is one standard deviation of the dara set.

The data set of from the Jerky time threshold benchmarking test can be seen in Figure **??**.

Figure 9.2: Jerky time threshold benchmarking process.

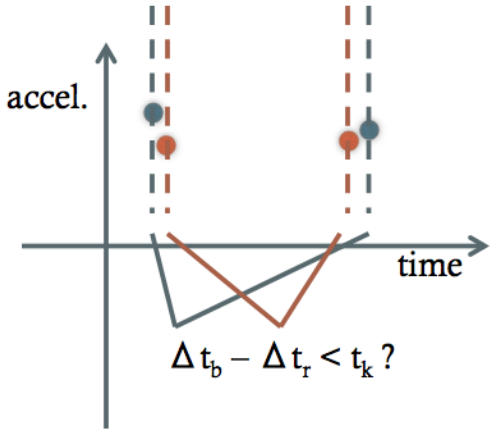
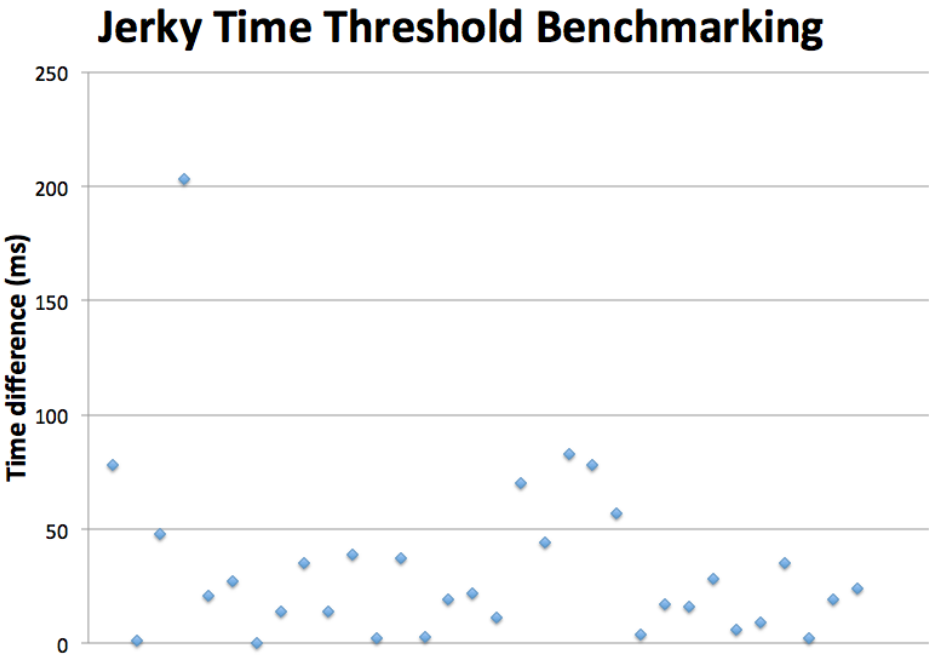


Figure 9.3: Jerky time threshold benchmarking result.



9.2 Jerky Experiments

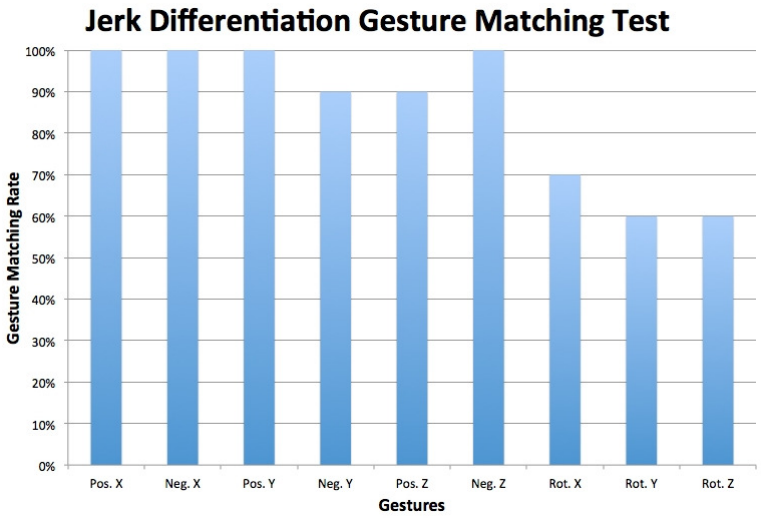
The Jerky experiments are identical to those done to test the Smoother end-user authentication protocol. Using the same two Android smartphones and gesture sets, tests were conducted to test Jerky's functionality. The first stage is the two phones are shaken together in one hand for a few seconds. Once each Jerky instance detects n_k jerk points, the set of time differences between each pair of adjacent jerk points is sent to the other phone. The third stage performs gesture comparison based upon the timing values in relation to the timing threshold, t_k. If the gesture parameters meet the conditions as determined by $\phi(g_i, g2_i)$, then the two users can communicate over P2P Bluetooth, otherwise, Bluetooth communication is suspended and the application aborts.

The same Smoother test plan was used with ten trials of each of the six linear gestures and three rotational gestures. The results of the described tests can be graphically seen in Figure 9.4. The Jerky protocol was able to correctly identify 100% of four of the six linear gestures. A similar relatively lower rotational gesture matching rate was also observed. The false negative rate is the reciprocal of the gesture matching rate.

The second experiment evaluated the false positive rate of each of the three protocols. I defined a false positive as a gesture match occurring between two gestures that do not exhibit the same motion but have similar acceleration and timing characteristics. In this experiment, two users each moved their phone with no knowledge of the motion path and timing of the other users gesture. I predicted that the library size of possible gestures would be large enough to result in a near zero false positive rate. After running fifty trials, the false positive rate was 0%. There were no incidental matches between two randomly selected gestures by the two users, same as the Smoother protocol.

The third test compared the speed entry of manually entered 4-digit PINs versus the average

Figure 9.4: Jerk gesture matching results.



speed of gesture entry for the Jerky end-user authentication protocol. Recall that I wrote an Android application that generates a random 4-digit number and measures the total time it takes of the user to type it in on the smartphone touchscreen. An average PIN takes 1.804 +/- 0.301 seconds to input on a smartphone touchscreen and an average gesture with the Jerk Determination takes 1.611 +/- 0.189 seconds. The difference is subtle but the Jerky gesture protocol takes slightly less time to acquire the necessary amount of acceleration data to generate a PIN.

Chapter 10

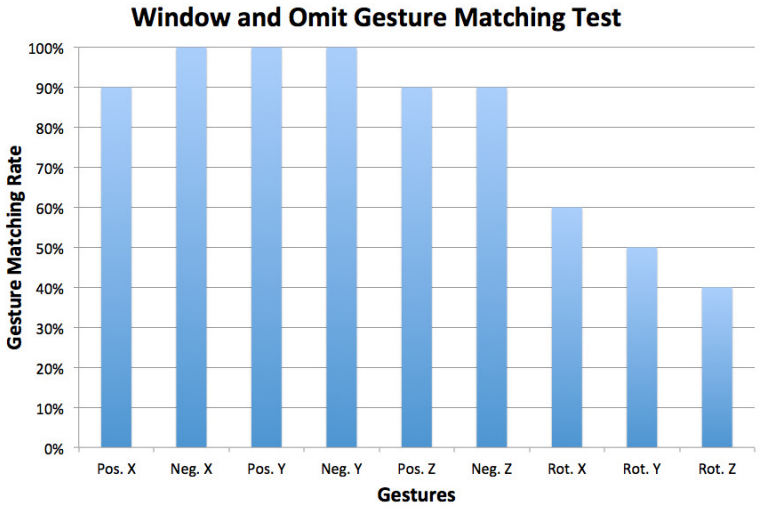
Window and Omit Test Plan and Results

10.1 Window and Omit Experiments

The Window and Omit experiments are identical to those done to test the Smoother and Jerky end-user authentication protocols. Using the same two Android smartphones and gestures sets, tests were conducted to test Window and Omit's functionality. The sequence of events for the test follows the same sequence of events of the Jerky protocol for the gesture aqcuisition stage. Recall that the Window and Omit protocol considers two gestures as similar if none of the data subsets are omitted during the gesture comparison stage.

The same Smoother and Jerky test plan was used with ten trials of each of the six linear gestures and three rotational gestures. The results of the gesture matching test can be graphically seen in Figure 10.1. The Window and Omit protocol was able to correctly gesture match half of the linear gestures 100% of the time and the other half 90% of the

Figure 10.1: Window and Omit gesture matching results.



time. The gesture matching rate is slightly lower than that of the Smoother and Jerky protocols, but the relative security of Window and Omit is significantly higher, which is discussed later.

The second experiment evaluated the false positive rate of each of the three protocols. The definition of a false positive is the same in this context, and as expected, there were no incidents of two gestures wrongfully matched.

Because the gesture acquisition stage of the Window and Omit protocol is the same as the Jerky protocol, the speed test results are the same.

Chapter 11

Discussion

11.1 Addressing Late Arrival Detection Challenges

11.1.1 Motion-based Sensor Power Cycling

Rather than immediately turning on the GPS to attempt to acquire a location fix, we took a more power conservative approach. When the application is sleeping, power consumption is at a minimum. Once the application wakes up, the accelerometer is activated to determine if the user is moving before the GPS is turned on to begin outdoor localization. The accelerometer provides data on the changes in the phone's acceleration, and we equated roughly twenty seconds of continual accelerometer updates as evidence of the user walking around, assuming that he/she is heading towards the destination by exiting the building. Once the accelerometer detects these physical changes, the data is stored in the cyber application so that the decision tree can turn off the accelerometer and then enable GPS to establish an initial outdoor location fix.

I used the accelerometer in this manner to shorten the time we needed the GPS to establish an initial location fix. By default, the GPS on the smartphone does not have a timeout, meaning, if the phone's GPS is on and the phone is sitting on a desk somewhere inside a building, the phone will continually attempt to connect to a satellite to establish a location fix. Since this operation is costly in terms of power consumption, delaying activation of the physical sensors, such as GPS and accelerometers, until the current time is within five minutes of the appointment time stored in the cyber model, can substantially reduce power consumption.

11.1.2 Arrival Detection with Bluetooth

The second challenge involves the difficulty in providing accurate indoor localization down to a specific room without using preconfigured, external hardware. I used the Bluetooth adapter already built into the smartphone to achieve relative indoor localization of the user. By discovering recognized Bluetooth devices in the nearby area based upon device name and MAC address, we're alleviated from the need of precise localization, which is hard to do without costly additional hardware. When the adapter sees other Bluetooth devices, the decision tree determines if they are friends/co-workers.

11.1.3 Detecting Outdoor-to-Indoor Transitions

Previously, in Section 2.3.3 we discussed the challenge of seamless indoor/outdoor localization. The decision tree transitions the state-machine to State 3 by enabling the smartphone's GPS after roughly twenty seconds of movement captured by the accelerometer. GPS serves two purposes once it is activated. First, GPS can effectively localize targets when outdoors. Again, we assume that the user has to walk outside in order to reach the destination, so once he/she steps outside, GPS can establish a position lock and provide position updates

periodically.

The second function of the GPS is to provide the decision tree with the point in time when the user leaves the outdoors and enters the destination building. I accomplishted this by using a timeout counter, which is reset every time the smartphone receives a position update from the satellite. The two scenarios of when the phone would cease to receive these notifications is either when the phone is no longer outside or if the user stops walking.

In both of these events, the timeout counter would reach zero, thus signaling the decision tree to disable GPS and transition to State 4 by enabling the Bluetooth adapter, as seen in Figure 4.2. This scheme provides a seamless transition from outdoor localization to indoor localization without any manual input from the user and without the user even knowing.

11.1.4 Estimating User Arrival Time without Location Initial Fix

In Section 2.3.4 we discussed the challenge of estimating a user's arrival time without a location fix. Recall from the formal model that α is a flag that indicates whether or not arriving at the meeting destination is sufficient to have arrived at the meeting or if the attendance requires reaching a specific room and ϵ is a time allowance to walk from the entrance of a building to a meeting room. Since we assume that these two values are negligible, PENDING is still able to provide a relative estimated time of arrival even though the user has moved indoors and the location fix is lost. By providing a final automatic notification to the contact upon the user's arrival to the building, the contact will have a more accurate estimate of how long it will take the user to reach the meeting room, in comparison to the notification sent while the user was walking outside, since the contact is already familiar with the layout of the building. The environmental shift from outdoors to indoors causes the algorithm to adapt to the change and automically send an SMS with fresh lateness information.

11.1.5 PENDING Implementation Challenges & Solutions

The four challenges pertaining to late arrival detection with automatic notification described in Section 2.3 were addressed and solved both in theory and practice. The research discussed here and the PENDING solution were prototyped in an Android application, which created several other challenges during the implementation process. Careful attention to resource allocation and thread timing on the smartphone was necessary in order to make the lateness detection algorithm work. Another challenge was the granularity at which the user could be aware of different state transitions and decisions made. I implemented clear visual cues for the user's benefit whenever there was a state change or a need for user interaction to complete a task.

11.1.6 Analysis of PENDING's Experimental Results

Considering the results from all the experiments, PENDING's localization power consumption is sufficiently less that any other localizaiton approach and PENDING's consistency in detecting late arrivals and new destinations is very satisfactory. PENDING's localization battery usage was only 30%, where as the battery usage for pure GPS and pure Bluetooth localization was more than double. PENDING also exhibited a 95% success rate at detecting a user's lateness and then send automatic notification to the contact at the destination. PENDING also has a 100% success rate at seamlessly switching between outdoor and indoor localization during isolated tests as well as detecting new destinations and updating the cyber-model look-up table for future use.

Even though GPS is a useful hardware tool for localization, using GPS for long durations of time on a mobile device consumes a considerable amount of power and shortens the life of the phone. With PENDING, we trim down the impact of GPS by implementing the

accelerometer-based algorithm for enabling GPS discussed in Section 4.1.2.

The pure Bluetooth indoor localization approach required the most battery power to discover other Bluetooth devices in the area. Walking around with the phone continually discovering devices consumes a great amount of power, but PENDING uses this feature sparingly to determine accurate user arrival to the destination.

This data is important because not only does it show promising results in addressing the challenges of power-efficient cyber-physical user lateness detection with automatic notification, but it also shows how the physical environment can induce adaptations in a useful algorithm on smartphones. With PENDING, we were able to reduce power consumption during user localization, save other peoples' time by being able to alert them of the user's lateness and arrival to the destination and allow for new destinations to be later used for user lateness detection.

11.2 Addressing End-User Authentication Challenges

11.2.1 Analysis of Gesture Protocol Experimental Results

From my experiments I learned several things about the four protocols I designed and implemented: (1) certain gestures are easier to recognize than others, especially when comparing linear gestures versus rotational gestures, (2) false positive rates become nonexistant when the threshold is small enough, (3) gesture-based end-user authentication protocols are secure ways to establish unique identity. The provided data gives valuable insight into the use of gestures to verify a user's identity in a P2P Bluetooth authentication session on smartphones.

11.2.2 Analysis of Bluehand's Security

Bluehand's security can be explained by evaluating two equations. The first equation represents the probability that a malicious user would correctly guess the gesture and threshold level. Recall that before any acceleration data is shared, the two users must select the same gesture and security level before proceeding on. There were sixteen different gestures we considered, so there is a 1/16 chance that the attacker would correctly guess the gesture being used. I also used three different security levels, so there is a 1/3 chance that the attacker would correctly guess that parameter. The chance that the attacker would correctly guess the gesture and security level is seen below:

$$P(g+t) = P(g) * P(t) = 0.0625 * 0.334 = 0.021 \tag{11.1}$$

A malicious user has a 2.1% chance of correctly guessing both the gesture and threshold level at the same time. The second equation calculates the probability that the user would be able pair up after correctly guessing the gesture and threshold. Recall the as the security level increases, the gesture recognition threshold becomes more stringent and requires increasing similarity between the two users' movements. The probability that an attacker would be able to pair up with another user is shown below:

$$P(B) = \begin{cases} P(g+t) * P(U) = 0.021 * 0.99 = 0.0207 \\ P(g+t) * P(S) = 0.021 * 0.958 = 0.0201 \\ P(g+t) * P(TS) = 0.021 * 0.954 = 0.02 \end{cases} \tag{11.2}$$

At the unclassified security level, the attacker has a 2.07% chance of pairing up with another user. At the secret security level, the attacker has a 2.01% chance of pairing up with the

other user and at the top security level, the probability is only 2.00%.

While the difference in probability of successful pairing between each security level is relatively small, the greater effect of the security levels is seen in the first equation. Including a second parameter to choose from great decreases the chances of an attacker from correctly guessing both the gesture and the security level.

Further Bluehand Security Opportunities To increase the Bluehand's security even further, we consider an end-user authentication protocol that requires four gestures strung together in sequential order to generate the equivalent of a 16-digit Bluetooth PIN, which is a more secure, but infrequently used option in conventional Bluetooth end-user authentication on numeric input devices (e.g., smartphones) [26].

To extrapolate our speed test results and the calculations above, we can say that a four-gesture authentication scheme would be faster than manually inputting a 16-digit PIN on a smartphone touchscreen and four times as secure as a single-gesture authentication scheme, resulting in a 0.05% probability that an attacker would correctly guess all four gestures in the correct order.

11.2.3 Analysis of Smoother, Jerky, and Window and Omit's Security

Through a series of calculations, I compared the relative security of various key spaces for each of the three end-user authentication protocols versus that of a traditional 4-digit PIN. The key space for a 4-digit pin is ten thousand possible combinations with a probability of a one-time successful brute force end-user masquerade attack as:

$$\frac{1}{(k_1 * k_2 * k_3 * k_4)} = \frac{1}{10 * 10 * 10 * 10} = \frac{1}{10000} \tag{11.3}$$

where k_i is the number of possible combinations for the ith digit of the PIN. To evaluate the protocols relative security, I make some assumptions. The first assumption is that the gesture is linear in nature. The second assumption is that the gesture is a series of five smaller linear movements, which was the case for the Smoother, Jerky, and Window and Omit experiments. The third assumption is that the user has the six linear gestures to choose from to string together in a series, even though there is a theoretical infinite number of linear motions in three-dimensional space.

Based upon these assumptions, the library of possible gesture combinations is:

$$g_1 * g_2 * g_3 * g_4 * g_5 = 6^5 = 7776 \tag{11.4}$$

where g_i is the number of possible linear motions available to the user to string together. Even though the key space for the gesture protocols is fewer than a traditional PIN, the probability of a one-time successful brute force end-user masquerade attack is not the multiplicative inverse of the value. The gesture timing characteristics have significant weight on the relative security of the protocols. In the Jerky protocol, if a pair of jerk points from two different gestures fall within 300 ms of one another, then the algorithm identifies them as similar. The average gesture entry speed for the protocol was 1.611 seconds, thus the average probability of a random pair of jerk points to be considered similar is $0.300/1.611 = 18.62\%$. I generalize the probability of a successful one-time brute force end-user masquerade attack against the Jerky protocol as:

$$\frac{1}{7776} * 0.1862^5 = 2.88 * 10^{-8} \tag{11.5}$$

The smoother protocol exhibits a similar representation of its relative security, where the

probability of a random pair of jerk points to be considered similar is 0.300 / 1.766 = 16.99%. I generalize the probability of a successful one-time brute force end-user masquerade attack against the Smoother protocol as:

$$\frac{1}{7776} * 0.1699^5 = 1.82 * 10^{-8} \tag{11.6}$$

Finally, the Window and Omit protocol considers two gestures as similar if their jerk pairs pass all nine subset checks. The probability that a subset passes decreases as the window count increases. Recall that a subset is omitted if any one of its windows to not meet the protocols gesture matching requirements. I generalize the probability of a successful one-time brute force end-user masquerade attack against the Window and Omit protocol as:

$$\frac{1}{7776} * \frac{1}{2} * \frac{1}{3} * \frac{1}{4} * ... * \frac{1}{10} = 3.54 * 10^{-11} \tag{11.7}$$

Based upon my calculations, it is evident that the probability of a successful random gesture mimic is near zero. Increasing the size of the gesture key space increases the relative security at an exponential rate. For example, by extrapolation, a 3-second string of gestures would nearly double the number of linear motions the user could execute based upon the results from our entry speed test. Thus, the library of possible combinations for a 3-second PIN can be equated as $6^10 = 60,466,176$.

Chapter 12

Conclusions

This thesis covers two specific problem domains of cyber-physical algorithms for enhancing collaboration, late user arrival detection and ad-hoc secure P2P pairing protocols. Late user arrival detection is a problem domain in collaborations planned in advanced and pairing protocols are unplanned and dynamic collaborations.

Late arrivals at meetings and jobs can have a number of substantial negative impacts. For example, on a construction site, construction activities are typically dependent upon one another and the late arrival of a contractor can leave workers idle and lead to cost overruns. The negative impact of delays or late arrivals can be mitigated or minimized if late arrivals can be predicted in advanced and affected parties notified.

Smartphones are a promising platform for detecting late arrivals since they are always carried by their users, have access to a user's calendar, and have a variety of sensors, such as GPS, that can be used for localization. A key challenge, however, is that localization approaches require significant power and can quickly exhaust a smartphone's battery. Another key challenge is indoor localization to a specific room. This paper presented an approach, called

PENDING, that intelligently power cycles localization sensors based on motion and utilizes Bluetooth beacons as a lightweight indoor localization mechanism for detecting arrivals in meeting rooms, estimates the user's arrival time to a destination and provides contacts with late arrival updates. The empirical results show that PENDING consumes 43% less power than late arrival detection with existing approaches and a 90% success rate at detecting late arrivals and automatically notifying affected parties.

Insecure collaborative communication can have a number of substantial negative impacts. For example, in a disaster relief site, medical personnel share the personal information of victims they are treating and an insecure method of pairing could result in a malicious user obtaining that information. Some of this negative impact can be mitigated, however, if end-user authentication could be quickly and securely established before collaboration.

Using motion gestures on smartphones is a promising technique for providing secure, ubiquitous P2P Bluetooth end-user authentication. A key challenge, however, is that there is a substantial trade-off between security and usability of a gesture-based authentication protocol. I presented collection of approaches, called Bluehand, Jerky, Smoother and WIndow and Omit, that effectively smoothens accelerometer data to make it more usuable for gesture comparision to authenticate a user and also gives the users security options to control how secure the session will be. The empirical results show that this combination of approaches provides an upwards of 100% linear gesture recognition rate at the secret security level and a 0% false positive rate. The results also show that gesture-based end-user authentication is faster than manually inputting a PIN on a smartphone touchscreen, and that by stringing multiple gestures together greatly increases the security quality of Bluehand.

Chapter 13

Appendix A

Appendix A contains the source code for the Smoother end-user authentication Android application discussed earlier. Figure 13.1 is an image of the file structure in Eclipse necessary to run and install the application onto an Android smartphone. For convenience, the sections and subsections headings of this appendix refer to folders and filenames, respectively.

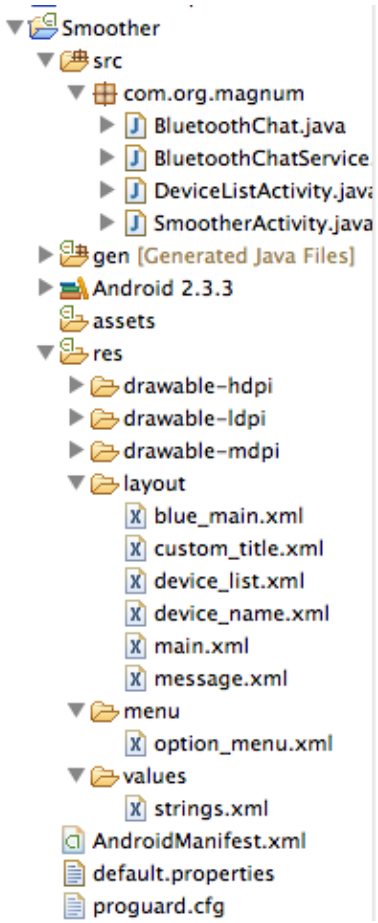
13.1 Smoother: src

13.1.1 Smoother: src/SmootherActivity.java

```
package com.org.magnum;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
```

Figure 13.1: Smoother file structure.




```
import android.util.FloatMath;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class SmootherActivity extends Activity implements
                        SensorEventListener, OnClickListener{

        Sensor mAccelerometer_;
        SensorManager mSensorManager_;

        TextView peakCountTextView_, elapsedTimeTextView_;
```

```
    Float accelVectorFloat_;
    Float[] accelVectorFloatArray_;

    long startTime_, currentTime_, elapsedTime_;

    Button resetButton_;

    int peakCount_;

    Intent bluetoothIntent_;

    BluetoothAdapter mBluetoothAdapter_;

    /** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    this.setRequestedOrientation
            (ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

    elapsedTimeTextView_ =
            (TextView) findViewById(R.id.elapsedTimeTextView);
    peakCountTextView_ =
            (TextView) findViewById(R.id.peakCountTextView);

    mSensorManager_ = (SensorManager)getSystemService
            (SENSOR_SERVICE);
    mAccelerometer_ =
            mSensorManager_.getDefaultSensor
            (Sensor.TYPE_ACCELEROMETER);

    mBluetoothAdapter_ = BluetoothAdapter.getDefaultAdapter();
    mBluetoothAdapter_.enable();

    accelVectorFloatArray_ = new Float[7];

    peakCount_ = 0;

    accelVectorFloatArray_[0] = (float) 0;
    accelVectorFloatArray_[1] = (float) 0;
```

```
        accelVectorFloatArray_[2] = (float) 0;
        accelVectorFloatArray_[3] = (float) 0;
        accelVectorFloatArray_[4] = (float) 0;
        accelVectorFloatArray_[5] = (float) 0;
        accelVectorFloatArray_[6] = (float) 0;

        resetButton_ = (Button) findViewById(R.id.resetButton);
        resetButton_.setOnClickListener(this);

        bluetoothIntent_ = new Intent(this, BluetoothChat.class);

        startTime_ = System.currentTimeMillis();

        mSensorManager_.registerListener(this,
                mAccelerometer_, SensorManager.SENSOR_DELAY_GAME);
    }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy)
         {
                // TODO Auto-generated method stub

        }

        @Override
        public void onSensorChanged(SensorEvent arg0) {
                // TODO Auto-generated method stub
                accelVectorFloat_ = FloatMath.sqrt(arg0.values[0]*
                                arg0.values[0] +
                                arg0.values[1]*arg0.values[1] +
                                arg0.values[2]*arg0.values[2]);

                accelVectorFloatArray_[0] =
                        accelVectorFloatArray_[1];
                accelVectorFloatArray_[1] =
                        accelVectorFloatArray_[2];
                accelVectorFloatArray_[2] =
                        accelVectorFloatArray_[3];
                accelVectorFloatArray_[3] =
                        accelVectorFloatArray_[4];
                accelVectorFloatArray_[4] =
                        accelVectorFloatArray_[5];
```

```
accelVectorFloatArray_[5] =
        accelVectorFloatArray_[6];
accelVectorFloatArray_[6] = accelVectorFloat_;

if((accelVectorFloatArray_[0] <
        accelVectorFloatArray_[1]) &&
   (accelVectorFloatArray_[1] <
        accelVectorFloatArray_[2]) &&
   (accelVectorFloatArray_[2] <
        accelVectorFloatArray_[3]) &&
   (accelVectorFloatArray_[3] >
        accelVectorFloatArray_[4]) &&
   (accelVectorFloatArray_[4] >
        accelVectorFloatArray_[5]) &&
   (accelVectorFloatArray_[5] >
        accelVectorFloatArray_[6]) &&
   (accelVectorFloatArray_[3] >
        (float) 15.00))
{
        peakCount_++;
        if(peakCount_ == 1){
                startTime_ =
                        System.
                        currentTimeMillis();
        }
        peakCountTextView_.
                setText(Integer.
                toString(peakCount_));
        if(peakCount_ == 5){
                mSensorManager_.
                        unregisterListener(this);
                currentTime_ =
                        System.
                        currentTimeMillis();
                elapsedTime_ = currentTime_
                        - startTime_;
                elapsedTimeTextView_.
                        setText(Long.toString(
                        elapsedTime_)
                        + " milliseconds");
                bluetoothIntent_.putExtra("TIME",
                        elapsedTime_);
```

```
                                startActivityForResult(
                                        bluetoothIntent_, 0);
                        }
                }
        }

        @Override
        public void onClick(View arg0) {
                // TODO Auto-generated method stub
                peakCount_ = 0;
                peakCountTextView_.setText("0");
                elapsedTimeTextView_.setText("0 milliseconds");
                mSensorManager_.registerListener(this,
                        mAccelerometer_,
                        SensorManager.SENSOR_DELAY_GAME);
                mBluetoothAdapter_.enable();
        }

        @Override
        public void onPause(){
                mSensorManager_.unregisterListener(this);
                super.onPause();
        }

        @Override
        public void onStop(){
                mSensorManager_.unregisterListener(this);
                mBluetoothAdapter_.disable();
                super.onStop();
        }
}
```

13.1.2 Smoother: src/BluetoothChat.java

```
package com.org.magnum;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
```

```
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class BluetoothChat extends Activity {
    // Debugging
    private static final String TAG = "BluetoothChat";
    private static final boolean D = true;

    // Message types sent from the BluetoothChatService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;

    // Key names received from the BluetoothChatService Handler
    public static final String DEVICE_NAME = "device_name";
    public static final String TOAST = "toast";

    // Intent request codes
    private static final int REQUEST_CONNECT_DEVICE = 1;
    private static final int REQUEST_ENABLE_BT = 2;

    // Layout Views
    private TextView mTitle;
    private ListView mConversationView;
    private EditText mOutEditText;
```

```
    private Button mSendButton;

    // Name of the connected device
    private String mConnectedDeviceName = null;
    // Array adapter for the conversation thread
    private ArrayAdapter<String> mConversationArrayAdapter;
    // String buffer for outgoing messages
    private StringBuffer mOutStringBuffer;
    // Local Bluetooth adapter
    private BluetoothAdapter mBluetoothAdapter = null;
    // Member object for the chat services
    private BluetoothChatService mChatService = null;

    Bundle extrasBundle_;
    long elapsedTime_;

    int timeThreshold_;
    int otherPIN_;

    String[] pinStringArray_;



    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if(D) Log.e(TAG, "+++ ON CREATE +++");

        extrasBundle_ = getIntent().getExtras();
        elapsedTime_ = extrasBundle_.getLong("TIME");

        timeThreshold_ = 309;

        pinStringArray_ = new String[3];

        // Set up the window layout
        requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
        setContentView(R.layout.blue_main);
        getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
                R.layout.custom_title);

        // Set up the custom title
```

```
        mTitle = (TextView) findViewById(R.id.title_left_text);
        mTitle.setText(R.string.app_name);
        mTitle = (TextView) findViewById(R.id.title_right_text);

        // Get local Bluetooth adapter
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        // If the adapter is null, then Bluetooth is not supported
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, "Bluetooth is not available",
                Toast.LENGTH_LONG).show();
            finish();
            return;
        }
    }

    @Override
    public void onStart() {
        super.onStart();
        if(D) Log.e(TAG, "++ ON START ++");

        // If BT is not on, request that it be enabled.
        // setupChat() will then be called during onActivityResult
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent =
                new Intent(BluetoothAdapter.
                        ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent,
                REQUEST_ENABLE_BT);
        // Otherwise, setup the chat session
        } else {
            if (mChatService == null) setupChat();
        }
    }

    @Override
    public synchronized void onResume() {
        super.onResume();
        if(D) Log.e(TAG, "+ ON RESUME +");

        // Performing this check in onResume()
        // covers the case in which BT was
```

```
        // not enabled during onStart(), so we
        // were paused to enable it...
        // onResume() will be called when
        // ACTION_REQUEST_ENABLE activity returns.
        if (mChatService != null) {
            // Only if the state is STATE_NONE, do we know that we
            // haven't started already
            if (mChatService.getState() ==
                BluetoothChatService.STATE_NONE) {
              // Start the Bluetooth chat services
              mChatService.start();
            }
        }
    }

    private void setupChat() {
        Log.d(TAG, "setupChat()");

        // Initialize the array adapter for the
        // conversation thread
        mConversationArrayAdapter =
                new ArrayAdapter<String>(this, R.layout.message);
        mConversationView = (ListView) findViewById(R.id.in);
        mConversationView.setAdapter(mConversationArrayAdapter);

        // Initialize the compose field with a listener
        // for the return key
        mOutEditText = (EditText) findViewById(
                R.id.edit_text_out);
        mOutEditText.setOnEditorActionListener(mWriteListener);

        // Initialize the send button with a listener
        // that for click events
        mSendButton = (Button) findViewById(R.id.button_send);

        mSendButton.setEnabled(false);

        mSendButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // Send a message using content of the
                // edit text widget
                TextView view = (TextView) findViewById(
```

```
                R.id.edit_text_out);
                String message = view.getText().toString();
                sendMessage(message);
            }
        });

        // Initialize the BluetoothChatService to
        // perform bluetooth connections
        mChatService = new BluetoothChatService(this, mHandler);

        // Initialize the buffer for outgoing messages
        mOutStringBuffer = new StringBuffer("");
    }

    @Override
    public synchronized void onPause() {
        super.onPause();
        if(D) Log.e(TAG, "- ON PAUSE -");
    }

    @Override
    public void onStop() {
        super.onStop();
        if(D) Log.e(TAG, "-- ON STOP --");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Stop the Bluetooth chat services
        if (mChatService != null) mChatService.stop();
        if(D) Log.e(TAG, "--- ON DESTROY ---");
    }

    private void ensureDiscoverable() {
        if(D) Log.d(TAG, "ensure discoverable");
        if (mBluetoothAdapter.getScanMode() !=
            BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
            Intent discoverableIntent =
                new Intent(BluetoothAdapter.
                ACTION_REQUEST_DISCOVERABLE);
            discoverableIntent.putExtra(
```

```
                BluetoothAdapter.
                        EXTRA_DISCOVERABLE_DURATION, 300);
            startActivity(discoverableIntent);
        }
    }

    /**
     * Sends a message.
     * @param message  A string of text to send.
     */
    private void sendMessage(String message) {
        // Check that we're actually connected
        // before trying anything
        if (mChatService.getState() !=
                BluetoothChatService.STATE_CONNECTED) {
            Toast.makeText(this, R.string.not_connected,
                Toast.LENGTH_SHORT).show();
            return;
        }

        // Check that there's actually something to send
        if (message.length() > 0) {
            // Get the message bytes and tell the
            // BluetoothChatService to write
            byte[] send = message.getBytes();
            mChatService.write(send);

            // Reset out string buffer to zero and
            // clear the edit text field
            mOutStringBuffer.setLength(0);
            mOutEditText.setText(mOutStringBuffer);
        }
    }

    // The action listener for the EditText widget,
    // to listen for the return key
    private TextView.OnEditorActionListener mWriteListener =
        new TextView.OnEditorActionListener() {
        public boolean onEditorAction(TextView view,
                int actionId, KeyEvent event) {
            // If the action is a key-up event on the return key,
            // send the message
```

```
            if (actionId == EditorInfo.IME_NULL &&
                event.getAction() ==
                KeyEvent.ACTION_UP) {
                String message = view.getText().toString();
                sendMessage(message);
            }
            if(D) Log.i(TAG, "END onEditorAction");
            return true;
        }
    };

    // The Handler that gets information back
    // from the BluetoothChatService
    private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case MESSAGE_STATE_CHANGE:
                if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: "
                 + msg.arg1);
                switch (msg.arg1) {
                case BluetoothChatService.STATE_CONNECTED:
                    mTitle.setText(R.string.title_connected_to);
                    mTitle.append(mConnectedDeviceName);
                    mConversationArrayAdapter.clear();
                    BluetoothChat.this.sendMessage("2nd PIN: " +
                        Long.toString(elapsedTime_));
                    break;
                case BluetoothChatService.STATE_CONNECTING:
                    mTitle.setText(R.string.title_connecting);
                    break;
                case BluetoothChatService.STATE_LISTEN:
                case BluetoothChatService.STATE_NONE:
                    mTitle.setText(R.string.title_not_connected);
                    break;
                }
                break;
            case MESSAGE_WRITE:
                byte[] writeBuf = (byte[]) msg.obj;
                // construct a string from the buffer
                String writeMessage = new String(writeBuf);
                mConversationArrayAdapter.add(
```

```
                "Me:  " + writeMessage);
            break;
        case MESSAGE_READ:
            byte[] readBuf = (byte[]) msg.obj;
            // construct a string from the valid
            // bytes in the buffer
            String readMessage = new String(
                    readBuf, 0, msg.arg1);
            pinStringArray_ = readMessage.split(" ");
            otherPIN_ = Integer.valueOf(pinStringArray_[2]);
            if(Math.abs(otherPIN_ - elapsedTime_) <=
                    timeThreshold_){
                    BluetoothChat.this.mSendButton.
                            setEnabled(true);
                    mConversationArrayAdapter.add(
                            mConnectedDeviceName+":  " +
                            readMessage +
                            ", PIN match");
            }
            else{
                    mConversationArrayAdapter.add(
                            mConnectedDeviceName+":  " +
                            readMessage +
                            ", PIN mismatch");
                    mBluetoothAdapter.disable();
                    finishActivity(0);
            }
            break;
        case MESSAGE_DEVICE_NAME:
            // save the connected device's name
            mConnectedDeviceName = msg.getData().
                    getString(DEVICE_NAME);
            Toast.makeText(getApplicationContext(),
                    "Connected to "
                   + mConnectedDeviceName,
                    Toast.LENGTH_SHORT).show();
            break;
        case MESSAGE_TOAST:
            Toast.makeText(getApplicationContext(),
                    msg.getData().getString(TOAST),
                   Toast.LENGTH_SHORT).show();
            break;
```

```
            }
        }
    };

    public void onActivityResult(int requestCode,
        int resultCode, Intent data) {
        if(D) Log.d(TAG, "onActivityResult " + resultCode);
        switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:
            // When DeviceListActivity returns with a
            // device to connect
            if (resultCode == Activity.RESULT_OK) {
                // Get the device MAC address
                String address = data.getExtras()
                        .getString(DeviceListActivity.
                        EXTRA_DEVICE_ADDRESS);
                // Get the BLuetoothDevice object
                BluetoothDevice device =
                        mBluetoothAdapter.
                        getRemoteDevice(address);
                // Attempt to connect to the device
                mChatService.connect(device);
            }
            break;
        case REQUEST_ENABLE_BT:
            // When the request to enable Bluetooth returns
            if (resultCode == Activity.RESULT_OK) {
                // Bluetooth is now enabled, so set
                // up a chat session
                setupChat();
            } else {
                // User did not enable Bluetooth or an
                // error occured
                Log.d(TAG, "BT not enabled");
                Toast.makeText(this, R.string.
                        bt_not_enabled_leaving,
                        Toast.LENGTH_SHORT).show();
                finish();
            }
        }
    }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.option_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.scan:
            // Launch the DeviceListActivity to see devices
            // and do scan
            Intent serverIntent = new Intent(this,
                DeviceListActivity.class);
            startActivityForResult(serverIntent,
                REQUEST_CONNECT_DEVICE);
            return true;
        case R.id.discoverable:
            // Ensure this device is discoverable by others
            ensureDiscoverable();
            return true;
        }
        return false;
    }
}
```

13.1.3 Smoother: src/BluetoothChatService.java

```
package com.org.magnum;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
```

```
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;


public class BluetoothChatService {
    // Debugging
    private static final String TAG = "BluetoothChatService";
    private static final boolean D = true;

    // Name for the SDP record when creating server socket
    private static final String NAME = "BluetoothChat";

    // Unique UUID for this application
    private static final UUID MY_UUID =
        UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");

    // Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0;
        // we're doing nothing
    public static final int STATE_LISTEN = 1;
        // now listening for incoming connections
    public static final int STATE_CONNECTING = 2;
        // now initiating an outgoing connection
    public static final int STATE_CONNECTED = 3;
        // now connected to a remote device

    /**
     * Constructor. Prepares a new BluetoothChat session.
     * @param context  The UI Activity Context
     * @param handler  A Handler to send messages
     * back to the UI Activity
     */
```

```
public BluetoothChatService(Context context,
    Handler handler) {
    mAdapter = BluetoothAdapter.getDefaultAdapter();
    mState = STATE_NONE;
    mHandler = handler;
}

/**
 * Set the current state of the chat connection
 * @param state  An integer defining the current
 * connection state
 */
private synchronized void setState
            (int state) {
    if (D) Log.d(TAG, "setState() " + mState +
            " -> " + state);
    mState = state;

    // Give the new state to the Handler so the UI
    // Activity can update
    mHandler.obtainMessage(BluetoothChat.
            MESSAGE_STATE_CHANGE,
            state, -1).sendToTarget();
}

/**
 * Return the current connection state. */
public synchronized int getState() {
    return mState;
}

/**
 * Start the chat service. Specifically
 * start AcceptThread to begin a
 * session in listening (server) mode.
    Called by the Activity onResume() */
public synchronized void start() {
    if (D) Log.d(TAG, "start");

    // Cancel any thread attempting to make a connection
    if (mConnectThread != null) {mConnectThread.cancel();
            mConnectThread = null;}
```

```
        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}

        // Start the thread to listen on a BluetoothServerSocket
        if (mAcceptThread == null) {
            mAcceptThread = new AcceptThread();
            mAcceptThread.start();
        }
        setState(STATE_LISTEN);
    }

    /**
     * Start the ConnectThread to initiate a
     * connection to a remote device.
     * @param device  The BluetoothDevice to connect
     */
    public synchronized void connect(BluetoothDevice device) {
        if (D) Log.d(TAG, "connect to: " + device);

        // Cancel any thread attempting to make a connection
        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {mConnectThread.cancel();
                mConnectThread = null;}
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}

        // Start the thread to connect with the given device
        mConnectThread = new ConnectThread(device);
        mConnectThread.start();
        setState(STATE_CONNECTING);
    }

    /**
     * Start the ConnectedThread to begin
     * managing a Bluetooth connection
     * @param socket  The BluetoothSocket on
     * which the connection was made
```

```
     * @param device  The BluetoothDevice that
     * has been connected
     */
    public synchronized void connected(BluetoothSocket socket,
        BluetoothDevice device) {
        if (D) Log.d(TAG, "connected");

        // Cancel the thread that completed the connection
        if (mConnectThread != null) {mConnectThread.cancel();
                mConnectThread = null;}

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}

        // Cancel the accept thread because we only want to
        // connect to one device
        if (mAcceptThread != null) {mAcceptThread.cancel();
                mAcceptThread = null;}

        // Start the thread to manage the connection
        // and perform transmissions
        mConnectedThread = new ConnectedThread(socket);
        mConnectedThread.start();

        // Send the name of the connected device
        // back to the UI Activity
        Message msg = mHandler.obtainMessage(
                BluetoothChat.MESSAGE_DEVICE_NAME);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothChat.DEVICE_NAME,
                device.getName());
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        setState(STATE_CONNECTED);
    }

    /**
     * Stop all threads
     */
    public synchronized void stop() {
```

```
        if (D) Log.d(TAG, "stop");
        if (mConnectThread != null) {mConnectThread.cancel();
                mConnectThread = null;}
        if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}
        if (mAcceptThread != null) {mAcceptThread.cancel();
                mAcceptThread = null;}
        setState(STATE_NONE);
    }

    /**
     * Write to the ConnectedThread in an unsynchronized manner
     * @param out The bytes to write
     * @see ConnectedThread#write(byte[])
     */
    public void write(byte[] out) {
        // Create temporary object
        ConnectedThread r;
        // Synchronize a copy of the ConnectedThread
        synchronized (this) {
            if (mState != STATE_CONNECTED) return;
            r = mConnectedThread;
        }
        // Perform the write unsynchronized
        r.write(out);
    }

    /**
     * Indicate that the connection attempt failed
     * and notify the UI Activity.
     */
    private void connectionFailed() {
        setState(STATE_LISTEN);

        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(
                BluetoothChat.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothChat.TOAST,
                "Unable to connect device");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
```

```
    }

    /**
     * Indicate that the connection was lost
     * and notify the UI Activity.
     */
    private void connectionLost() {
        setState(STATE_LISTEN);

        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(
                BluetoothChat.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothChat.TOAST,
                "Device connection was lost");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
    }

    /**
     * This thread runs while listening for incoming connections.
     * It behaves
     * like a server-side client. It runs until a
     * connection is accepted
     * (or until cancelled).
     */
    private class AcceptThread extends Thread {
        // The local server socket
        private final BluetoothServerSocket mmServerSocket;

        public AcceptThread() {
            BluetoothServerSocket tmp = null;

            // Create a new listening server socket
            try {
                tmp = mAdapter.listenUsingRfcommWithServiceRecord(
                        NAME, MY_UUID);
            } catch (IOException e) {
                Log.e(TAG, "listen() failed", e);
            }
            mmServerSocket = tmp;
        }
```

```
public void run() {
    if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
    setName("AcceptThread");
    BluetoothSocket socket = null;

    // Listen to the server socket if we're not connected
    while (mState != STATE_CONNECTED) {
        try {
            // This is a blocking call and will
            // only return on a
            // successful connection or an exception
            socket = mmServerSocket.accept();
        } catch (IOException e) {
            Log.e(TAG, "accept() failed", e);
            break;
        }

        // If a connection was accepted
        if (socket != null) {
            synchronized (BluetoothChatService.this) {
                switch (mState) {
                case STATE_LISTEN:
                case STATE_CONNECTING:
                    // Situation normal.
                    // Start the connected thread.
                    connected(socket,
                        socket.getRemoteDevice());
                    break;
                case STATE_NONE:
                case STATE_CONNECTED:
                    // Either not ready or
                    // already connected.
                    // Terminate new socket.
                    try {
                        socket.close();
                    } catch (IOException e) {
                        Log.e(TAG,
                                "Could not close
                                unwanted socket", e);
                    }
                    break;
```

```
                    }
                }
            }
        }
        if (D) Log.i(TAG, "END mAcceptThread");
    }

    public void cancel() {
        if (D) Log.d(TAG, "cancel " + this);
        try {
            mmServerSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "close() of server failed", e);
        }
    }
}


/**
 * This thread runs while attempting to
 * make an outgoing connection
 * with a device. It runs straight through;
 * the connection either
 * succeeds or fails.
 */
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        mmDevice = device;
        BluetoothSocket tmp = null;

        // Get a BluetoothSocket for a connection with the
        // given BluetoothDevice
        try {
            tmp = device.
                    createRfcommSocketToServiceRecord(
                    MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "create() failed", e);
        }
```

```
        mmSocket = tmp;
    }

    public void run() {
        Log.i(TAG, "BEGIN mConnectThread");
        setName("ConnectThread");

        // Always cancel discovery because
        //it will slow down a connection
        mAdapter.cancelDiscovery();

        // Make a connection to the BluetoothSocket
        try {
            // This is a blocking call and will only
            // return on a
            // successful connection or an exception
            mmSocket.connect();
        } catch (IOException e) {
            connectionFailed();
            // Close the socket
            try {
                mmSocket.close();
            } catch (IOException e2) {
                Log.e(TAG, "unable to close() socket
                    during connection failure", e2);
            }
            // Start the service over to restart
            // listening mode
            BluetoothChatService.this.start();
            return;
        }

        // Reset the ConnectThread because we're done
        synchronized (BluetoothChatService.this) {
            mConnectThread = null;
        }

        // Start the connected thread
        connected(mmSocket, mmDevice);
    }

    public void cancel() {
```

```
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }

    /**
     * This thread runs during a connection with a remote device.
     * It handles all incoming and outgoing transmissions.
     */
    private class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            Log.d(TAG, "create ConnectedThread");
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            // Get the BluetoothSocket input and output streams
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
                Log.e(TAG, "temp sockets not created", e);
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectedThread");
            byte[] buffer = new byte[1024];
            int bytes;

            // Keep listening to the InputStream while connected
            while (true) {
```

```
            try {
                // Read from the InputStream
                bytes = mmInStream.read(buffer);

                // Send the obtained bytes to the UI Activity
                mHandler.obtainMessage(BluetoothChat.
                    MESSAGE_READ,
                            bytes, -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {
                Log.e(TAG, "disconnected", e);
                connectionLost();
                break;
            }
        }
    }

    /**
     * Write to the connected OutStream.
     * @param buffer  The bytes to write
     */
    public void write(byte[] buffer) {
        try {
            mmOutStream.write(buffer);

            // Share the sent message back to the UI Activity
            mHandler.obtainMessage(BluetoothChat.
            MESSAGE_WRITE,
                    -1, -1, buffer)
                    .sendToTarget();
        } catch (IOException e) {
            Log.e(TAG, "Exception during write", e);
        }
    }

    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "close() of connect socket failed", e);
        }
    }
```

```
    }
}
```

13.1.4 Smoother: src/DeviceListActivity.java

```
package com.org.magnum;

import java.util.Set;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class DeviceListActivity extends Activity {
    // Debugging
    private static final String TAG = "DeviceListActivity";
    private static final boolean D = true;

    // Return Intent extra
    public static String EXTRA_DEVICE_ADDRESS = "device_address";

    // Member fields
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter<String> mPairedDevicesArrayAdapter;
    private ArrayAdapter<String> mNewDevicesArrayAdapter;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Setup the window
    requestWindowFeature(Window.
            FEATURE_INDETERMINATE_PROGRESS);
    setContentView(R.layout.device_list);

    // Set result CANCELED incase the user backs out
    setResult(Activity.RESULT_CANCELED);

    // Initialize the button to perform device discovery
    Button scanButton = (Button) findViewById(
            R.id.button_scan);
    scanButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            doDiscovery();
            v.setVisibility(View.GONE);
        }
    });

    // Initialize array adapters. One for already
    // paired devices and
    // one for newly discovered devices
    mPairedDevicesArrayAdapter =
            new ArrayAdapter<String>(this,
            R.layout.device_name);
    mNewDevicesArrayAdapter =
            new ArrayAdapter<String>(this,
            R.layout.device_name);

    // Find and set up the ListView for paired devices
    ListView pairedListView =
            (ListView) findViewById(R.id.paired_devices);
    pairedListView.setAdapter(mPairedDevicesArrayAdapter);
    pairedListView.setOnItemClickListener(
            mDeviceClickListener);

    // Find and set up the ListView for newly
    // discovered devices
    ListView newDevicesListView =
```

```
            (ListView) findViewById(R.id.new_devices);
        newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
        newDevicesListView.setOnItemClickListener(
                mDeviceClickListener);

        // Register for broadcasts when a device is discovered
        IntentFilter filter =
                new IntentFilter(BluetoothDevice.ACTION_FOUND);
        this.registerReceiver(mReceiver, filter);

        // Register for broadcasts when discovery has finished
        filter = new IntentFilter(
                BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        this.registerReceiver(mReceiver, filter);

        // Get the local Bluetooth adapter
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();

        // Get a set of currently paired devices
        Set<BluetoothDevice> pairedDevices =
                mBtAdapter.getBondedDevices();

        // If there are paired devices, add each
        // one to the ArrayAdapter
        if (pairedDevices.size() > 0) {
            findViewById(R.id.title_paired_devices).
                setVisibility(View.VISIBLE);
            for (BluetoothDevice device : pairedDevices) {
                mPairedDevicesArrayAdapter.add(device.getName() +
                "\n" + device.getAddress());
            }
        } else {
            String noDevices = getResources().
                getText(R.string.none_paired).toString();
            mPairedDevicesArrayAdapter.add(noDevices);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
```

```
        // Make sure we're not doing discovery anymore
        if (mBtAdapter != null) {
            mBtAdapter.cancelDiscovery();
        }

        // Unregister broadcast listeners
        this.unregisterReceiver(mReceiver);
    }

    /**
     * Start device discover with the BluetoothAdapter
     */
    private void doDiscovery() {
        if (D) Log.d(TAG, "doDiscovery()");

        // Indicate scanning in the title
        setProgressBarIndeterminateVisibility(true);
        setTitle(R.string.scanning);

        // Turn on sub-title for new devices
        findViewById(R.id.title_new_devices).
                setVisibility(View.VISIBLE);

        // If we're already discovering, stop it
        if (mBtAdapter.isDiscovering()) {
            mBtAdapter.cancelDiscovery();
        }

        // Request discover from BluetoothAdapter
        mBtAdapter.startDiscovery();
    }

    // The on-click listener for all devices in the ListViews
    private OnItemClickListener mDeviceClickListener =
        new OnItemClickListener() {
        public void onItemClick(AdapterView<?> av,
        View v, int arg2, long arg3) {
            // Cancel discovery because it's
            // costly and we're about to connect
            mBtAdapter.cancelDiscovery();

            // Get the device MAC address,
```

```
            // which is the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Create the result Intent and include
            // the MAC address
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

            // Set result and finish this Activity
            setResult(Activity.RESULT_OK, intent);
            finish();
        }
    };

    // The BroadcastReceiver that listens for
    // discovered devices and
    // changes the title when discovery is finished
    private final BroadcastReceiver mReceiver =
        new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            // When discovery finds a device
            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                // Get the BluetoothDevice object from the Intent
                BluetoothDevice device =
                        intent.getParcelableExtra(
                        BluetoothDevice.EXTRA_DEVICE);
                // If it's already paired, skip it,
                // because it's been listed already
                if (device.getBondState() !=
                        BluetoothDevice.BOND_BONDED) {
                    mNewDevicesArrayAdapter.add(
                        device.getName() + "\n" +
                        device.getAddress());
                }
            // When discovery is finished,
            // change the Activity title
            } else if (BluetoothAdapter.
                ACTION_DISCOVERY_FINISHED.equals(action)) {
```

```
                setProgressBarIndeterminateVisibility(false);
                setTitle(R.string.select_device);
                if (mNewDevicesArrayAdapter.getCount() == 0) {
                    String noDevices = getResources().
                        getText(R.string.none_found).toString();
                    mNewDevicesArrayAdapter.add(noDevices);
                }
            }
        }
    };
}
```

13.2 Smoother: res/layout

13.2.1 Smoother: res/layout/bluemain.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <ListView android:id="@+id/in"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:stackFromBottom="true"
        android:transcriptMode="alwaysScroll"
        android:layout_weight="1"
    />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        >
        <EditText android:id="@+id/edit_text_out"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:layout_gravity="bottom"
        />
```

```
        <Button android:id="@+id/button_send"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/send"
        />
    </LinearLayout>
</LinearLayout>
```

13.2.2 Smoother: res/layout/customtitle.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:gravity="center_vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <TextView android:id="@+id/title_left_text"
      android:layout_alignParentLeft="true"
      android:ellipsize="end"
      android:singleLine="true"
      style="?android:attr/windowTitleStyle"
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:layout_weight="1"
    />
  <TextView android:id="@+id/title_right_text"
        android:layout_alignParentRight="true"
        android:ellipsize="end"
        android:singleLine="true"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:textColor="#fff"
        android:layout_weight="1"
    />
</RelativeLayout>
```

13.2.3 Smoother: res/layout/devicelist.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView android:id="@+id/title_paired_devices"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text="@string/title_paired_devices"
         android:visibility="gone"
         android:background="#666"
         android:textColor="#fff"
         android:paddingLeft="5dp"
     />
     <ListView android:id="@+id/paired_devices"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:stackFromBottom="true"
         android:layout_weight="1"
     />
     <TextView android:id="@+id/title_new_devices"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text="@string/title_other_devices"
         android:visibility="gone"
         android:background="#666"
         android:textColor="#fff"
         android:paddingLeft="5dp"
     />
     <ListView android:id="@+id/new_devices"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:stackFromBottom="true"
         android:layout_weight="2"
     />
     <Button android:id="@+id/button_scan"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text="@string/button_scan"
     />
</LinearLayout>
```

13.2.4 Smoother: res/layout/devicename.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
  xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:padding="5dp"
/>
```

13.2.5 Smoother: res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <ImageView android:src="@drawable/smoother"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:id="@+id/imageView1">
    </ImageView>
    <Button android:layout_gravity="center_horizontal"
                    android:text="Reset"
                    android:id="@+id/resetButton"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content">
    </Button>
    <LinearLayout android:layout_width="match_parent"
                    android:id="@+id/linearLayout1"
                    android:layout_height="wrap_content">
        <TextView android:layout_gravity="center_horizontal"
                            android:layout_weight="1"
                            android:id="@+id/textView1"
                            android:textAppearance=
                            "?android:attr/textAppearanceMedium"
                            android:text="Peak count: "
```

```
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content">
        </TextView>
        <TextView android:layout_gravity="center_horizontal"
                        android:layout_weight="1"
                        android:id="@+id/peakCountTextView"
                        android:textAppearance=
                        "?android:attr/textAppearanceMedium"
                        android:text="0"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content">
        </TextView>
    </LinearLayout>
    <LinearLayout android:layout_width="match_parent"
                          android:id="@+id/linearLayout2"
                          android:layout_height="wrap_content">
        <TextView android:layout_weight="1"
                        android:id="@+id/textView2"
                        android:textAppearance=
                        "?android:attr/textAppearanceMedium"
                        android:text="Elapsed Time: "
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content">
        </TextView>
        <TextView android:layout_weight="1"
                        android:id="@+id/elapsedTimeTextView"
                        android:textAppearance=
                        "?android:attr/textAppearanceMedium"
                        android:text="0 milliseconds"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content">
        </TextView>
    </LinearLayout>
</LinearLayout>
```

13.2.6 Smoother: /res/layout/message.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.
                android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:padding="5dp"
/>
```

13.3 Smoother: res/menu

13.3.1 Smoother: res/menu/optionmenu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/scan"
          android:icon="@android:drawable/ic_menu_search"
          android:title="@string/connect" />
    <item android:id="@+id/discoverable"
          android:icon="@android:drawable/ic_menu_mylocation"
          android:title="@string/discoverable" />
</menu>
```

13.4 Smoother: res/values

13.4.1 Smoother: res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Smoother</string>

    <!-- BluetoothChat -->
    <string name="send">Send</string>
    <string name="not_connected">You are not
        connected to a device</string>
    <string name="bt_not_enabled_leaving">
        Bluetooth was not enabled.
        Leaving Bluetooth Chat.</string>
    <string name="title_connecting">connecting...</string>
    <string name="title_connected_to">connected: </string>
    <string name="title_not_connected">not connected</string>
```

```
    <!--  DeviceListActivity -->
    <string name="scanning">scanning
        for devices...</string>
    <string name="select_device">select a
        device to connect</string>
    <string name="none_paired">No devices
        have been paired</string>
    <string name="none_found">No devices
        found</string>
    <string name="title_paired_devices">
        Paired Devices</string>
    <string name="title_other_devices">Other
        Available Devices</string>
    <string name="button_scan">Scan for
        devices</string>

    <!-- Options Menu -->
    <string name="connect">Connect a
        device</string>
    <string name="discoverable">Make
        discoverable</string>
</resources>
```

13.4.2 Smoother: res/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.
        android.com/apk/res/android"
      package="com.org.magnum"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />
    <uses-permission android:name=
        "android.permission.BLUETOOTH"></uses-permission>
    <uses-permission android:name=
        "android.permission.BLUETOOTH_ADMIN"></uses-permission>

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".SmootherActivity"
                  android:label="@string/app_name">
```

```
            <intent-filter>
                <action android:name=
                        "android.intent.action.MAIN" />
                <category android:name=
                        "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".BluetoothChat"
                  android:label="@string/app_name"
                  android:configChanges="orientation|
                        keyboardHidden"
        />
        <activity android:name=".DeviceListActivity"
                  android:label="@string/select_device"
                  android:theme="@android:style/Theme.Dialog"
                  android:configChanges="orientation|
                 keyboardHidden"
        />

    </application>
</manifest>
```

Chapter 14

Appendix B

Appendix B contains the source code for the Jerky end-user authentication Android application discussed earlier. Figure 14.1 is an image of the file structure in Eclipse necessary to run and install the application onto an Android smartphone. For convenience, the sections and subsections headings of this appendix refer to folders and filenames, respectively.

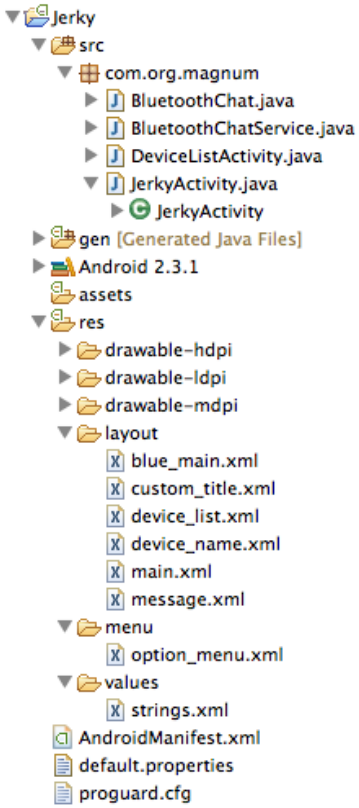
14.1 Jerky: src

14.1.1 Jerky: src/JerkyActivity.java

```
package com.org.magnum;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.FloatMath;
import android.view.View;
import android.view.View.OnClickListener;
```

Figure 14.1: Jerky file structure.



```
import android.widget.Button;
import android.widget.TextView;

public class JerkyActivity extends Activity implements
        SensorEventListener, OnClickListener{
    TextView jerkCountTextView_;
    Sensor mAccelerometer_;
    SensorManager mSensorManager_;
    Button resetButton_;

    int jerkCount_;
    double accelVector_, slopeArray_[], accelVectorArray_[],
        elapsedTime_, elapsedTimeArray_[], timeStampArray_[],
        jerkPointArray_[], timeDifferenceArray_[];

    long startTime_, currentTime_;
    Intent bluetoothIntent_;

    BluetoothAdapter mBluetoothAdapter_;

        /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        jerkCountTextView_ = (TextView) findViewById(
                R.id.jerkCountTextView);

        this.setRequestedOrientation(ActivityInfo.
                SCREEN_ORIENTATION_PORTRAIT);

        mSensorManager_ = (SensorManager)
                getSystemService(SENSOR_SERVICE);
        mAccelerometer_ = mSensorManager_.
                getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        resetButton_ = (Button) findViewById(R.id.resetButton);
        resetButton_.setOnClickListener(this);

        bluetoothIntent_ = new Intent(this, BluetoothChat.class);
```

```
        mBluetoothAdapter_ = BluetoothAdapter.getDefaultAdapter();
        mBluetoothAdapter_.enable();

        accelVectorArray_ = new double[3];
        for(int i = 0; i < accelVectorArray_.length; i++){
                accelVectorArray_[i] = 0.00;
        }

        slopeArray_ = new double[2];
        for(int j = 0; j < slopeArray_.length; j++){
                slopeArray_[j] = 0.00;
        }

        elapsedTimeArray_ = new double[3];
        for(int k = 0; k < elapsedTimeArray_.length; k++){
                elapsedTimeArray_[k] = 0.00;
        }

        timeStampArray_ = new double[5];
        for(int e = 0; e < timeStampArray_.length; e++){
                timeStampArray_[e] = 0.00;
        }

        jerkPointArray_ = new double[5];
        for(int v = 0; v < jerkPointArray_.length; v++){
                jerkPointArray_[v] = 0.00;
        }

        timeDifferenceArray_ = new double[4];
        for(int t = 0; t < timeDifferenceArray_.length; t++){
                timeDifferenceArray_[t] = 0.00;
        }

        jerkCount_ = 0;
        startTime_ = System.currentTimeMillis();
        mSensorManager_.registerListener(this, mAccelerometer_,
                SensorManager.SENSOR_DELAY_UI);

    }

        @Override
        public void onClick(View arg0) {
```

```
            // TODO Auto-generated method stub
            jerkCount_ = 0;
            jerkCountTextView_.setText(Integer.toString(
                    jerkCount_));
            startTime_ = System.currentTimeMillis();
    }

    @Override
    public void onAccuracyChanged(Sensor arg0, int arg1) {
            // TODO Auto-generated method stub

    }

    @Override
    public void onSensorChanged(SensorEvent event) {
            // TODO Auto-generated method stub
            accelVector_ = (double) FloatMath.sqrt(
                    event.values[0]*event.values[0] +
                    event.values[1]*event.values[1] +
                    event.values[2]*event.values[2]);

            accelVectorArray_[0] = accelVectorArray_[1];
            accelVectorArray_[1] = accelVectorArray_[2];
            accelVectorArray_[2] = accelVector_;

            currentTime_ = System.currentTimeMillis();
            elapsedTime_ = (((double) currentTime_) -
                    elapsedTimeArray_[1]) / 100;

            elapsedTimeArray_[0] = elapsedTimeArray_[1];
            elapsedTimeArray_[1] = elapsedTimeArray_[2];
            elapsedTimeArray_[2] = elapsedTime_;

            slopeArray_[0] = (accelVectorArray_[1] -
                    accelVectorArray_[0]) / (elapsedTimeArray_[1] -
                    elapsedTimeArray_[0]);
            slopeArray_[1] = (accelVectorArray_[2] -
                    accelVectorArray_[1]) / (elapsedTimeArray_[2] -
                    elapsedTimeArray_[1]);

            if((slopeArray_[0] > 0) && (slopeArray_[1] < 0)
                    && (accelVectorArray_[1] > 14.00)){
```

```
                        timeStampArray_[jerkCount_] =
                                elapsedTimeArray_[1];
                        jerkPointArray_[jerkCount_] =
                                accelVectorArray_[1];
                        jerkCount_++;
                }
                jerkCountTextView_.setText(Integer.
                        toString(jerkCount_));
                if(jerkCount_ == 5){
                        mSensorManager_.unregisterListener(this);
                        for(int q = 0; q < timeDifferenceArray_.
                                length; q++){
                                timeDifferenceArray_[q] =
                                (timeStampArray_[q+1] -
                                timeStampArray_[q]) * 100;
                        }

                        bluetoothIntent_.putExtra("TIMESTAMPS",
                                timeDifferenceArray_);
                        bluetoothIntent_.putExtra("JERKS",
                                jerkPointArray_);
                        startActivityForResult(bluetoothIntent_, 0);
                }
        }

        @Override
        public void onPause(){
                mSensorManager_.unregisterListener(this);
                super.onPause();
        }

        @Override
        public void onStop(){
                mSensorManager_.unregisterListener(this);
                super.onStop();
        }
}
```

14.1.2 Jerky: src/DeviceListActivity.java

The Jerky DeviceListActivity.java file is identical to that from Appendix A.

14.1.3 Jerky: src/BluetoothChatService.java

The Jerky BluetoothChatService.java file is identical to that from Appendix B.

14.1.4 Jerky: src/BluetoothChat.java

```
package com.org.magnum;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class BluetoothChat extends Activity {
    // Debugging
    private static final String TAG = "BluetoothChat";
    private static final boolean D = true;

    // Message types sent from the BluetoothChatService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
```

```
public static final int MESSAGE_DEVICE_NAME = 4;
public static final int MESSAGE_TOAST = 5;

// Key names received from the BluetoothChatService Handler
public static final String DEVICE_NAME = "device_name";
public static final String TOAST = "toast";

// Intent request codes
private static final int REQUEST_CONNECT_DEVICE = 1;
private static final int REQUEST_ENABLE_BT = 2;

// Layout Views
private TextView mTitle;
private ListView mConversationView;
private EditText mOutEditText;
private Button mSendButton;

// Name of the connected device
private String mConnectedDeviceName = null;
// Array adapter for the conversation thread
private ArrayAdapter<String> mConversationArrayAdapter;
// String buffer for outgoing messages
private StringBuffer mOutStringBuffer;
// Local Bluetooth adapter
private BluetoothAdapter mBluetoothAdapter = null;
// Member object for the chat services
private BluetoothChatService mChatService = null;

Bundle extrasBundle_;

double timeDifferenceArray_[], jerkPointArray_[],
    otherTimeDifferenceArray_[], otherJerkPointArray_[],
    timeThreshold_, jerkThreshold_;
String otherPhoneMessageArray_[];

boolean timePassed_, jerkPassed_;

Handler gestureCheckHandler_;
Runnable gestureCheckRunnable_;

String theMessage_;
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if(D) Log.e(TAG, "+++ ON CREATE +++");

    extrasBundle_ = getIntent().getExtras();
    //elapsedTime_ = extrasBundle_.getLong("TIME");

    timeDifferenceArray_ = new double[4];
    for(int i = 0; i < timeDifferenceArray_.length; i++){
            timeDifferenceArray_[i] = 0.00;
    }

    jerkPointArray_ = new double[5];
    for(int j = 0; j < jerkPointArray_.length; j++){
            jerkPointArray_[j] = 0.00;
    }

    otherTimeDifferenceArray_ = new double[4];
    for(int w = 0; w < otherTimeDifferenceArray_.length; w++){
            otherTimeDifferenceArray_[w] = 0.00;
    }

    otherJerkPointArray_ = new double[5];
    for(int r = 0; r < otherJerkPointArray_.length; r++){
            otherJerkPointArray_[r] = 0.00;
    }

    otherPhoneMessageArray_ = new String[10];

    timeDifferenceArray_ = extrasBundle_.getDoubleArray(
            "TIMESTAMPS");
    jerkPointArray_ = extrasBundle_.getDoubleArray("JERKS");

    timePassed_ = false;
    jerkPassed_ = false;

    timeThreshold_ = 102.00;
    jerkThreshold_ = 5.42;

    // Set up the window layout
    requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
```

```
setContentView(R.layout.blue_main);
getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
        R.layout.custom_title);

// Set up the custom title
mTitle = (TextView) findViewById(R.id.title_left_text);
mTitle.setText(R.string.app_name);
mTitle = (TextView) findViewById(R.id.title_right_text);

// Get local Bluetooth adapter
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// If the adapter is null, then Bluetooth is not supported
if (mBluetoothAdapter == null) {
    Toast.makeText(this, "Bluetooth is not available",
        Toast.LENGTH_LONG).show();
    finish();
    return;
}

gestureCheckHandler_ = new Handler();

gestureCheckRunnable_ = new Runnable(){
        public void run(){
                otherPhoneMessageArray_ = theMessage_.
                        split(" ");
                for(int g = 0; g < otherJerkPointArray_.
                        length; g++){
                        otherJerkPointArray_[g] = Double.
                                valueOf(
                                otherPhoneMessageArray_
                                [g+1]);
                        if(Math.abs(otherJerkPointArray_[g]
                         - jerkPointArray_[g]) <
                        jerkThreshold_){
                                jerkPassed_ = true;
                        }
                        else
                        {
                                jerkPassed_ = false;
                                g = otherJerkPointArray_.
                                        length;
```

```
                        }
                    }

                    for(int s = 0; s < otherTimeDifferenceArray_.
                        length; s++){
                        otherTimeDifferenceArray_[s] =
                        Double.valueOf(
                        otherPhoneMessageArray_[s+7]);
                        if(Math.abs(
                                otherTimeDifferenceArray_[s]
                         - timeDifferenceArray_[s]) <
                        timeThreshold_){
                                timePassed_ = true;
                        }
                        else
                        {
                                timePassed_ = false;
                                s = otherTimeDifferenceArray_.
                                        length;
                        }
                    }

            if((timeDifferenceArray_[0] != 0.00) &&
                    (timePassed_) && (jerkPassed_)){
                    BluetoothChat.this.mSendButton.
                            setEnabled(true);
                    mConversationArrayAdapter.add(
                    mConnectedDeviceName +
                    ": PIN match");
            }
            else{
                    mConversationArrayAdapter.add(
                    mConnectedDeviceName +
                    ": PIN mismatch");
            }
    };
}

@Override
public void onStart() {
    super.onStart();
    if(D) Log.e(TAG, "++ ON START ++");
```

```
        // If BT is not on, request that it be enabled.
        // setupChat() will then be called during onActivityResult
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent = new Intent(BluetoothAdapter.
                ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        // Otherwise, setup the chat session
        } else {
            if (mChatService == null) setupChat();
        }
    }

    @Override
    public synchronized void onResume() {
        super.onResume();
        if(D) Log.e(TAG, "+ ON RESUME +");

        // Performing this check in onResume()
        // covers the case in which BT was
        // not enabled during onStart(), so we were
        // paused to enable it...
        // onResume() will be called when ACTION_REQUEST_ENABLE
        // activity returns.
        if (mChatService != null) {
            // Only if the state is STATE_NONE, do we know
            // that we haven't started already
            if (mChatService.getState() == BluetoothChatService.
                STATE_NONE) {
              // Start the Bluetooth chat services
              mChatService.start();
            }
        }
    }

    private void setupChat() {
        Log.d(TAG, "setupChat()");

        // Initialize the array adapter for the conversation thread
        mConversationArrayAdapter = new ArrayAdapter<String>(this,
                R.layout.message);
        mConversationView = (ListView) findViewById(R.id.in);
```

```
        mConversationView.setAdapter(mConversationArrayAdapter);

        // Initialize the compose field with a listener for
        // the return key
        mOutEditText = (EditText) findViewById(R.id.edit_text_out);
        mOutEditText.setOnEditorActionListener(mWriteListener);

        // Initialize the send button with a listener
        // that for click events
        mSendButton = (Button) findViewById(R.id.button_send);

        mSendButton.setEnabled(false);

        mSendButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // Send a message using content of the edit
                // text widget
                TextView view = (TextView) findViewById(
                        R.id.edit_text_out);
                String message = view.getText().toString();
                sendMessage(message);
            }
        });

        // Initialize the BluetoothChatService to
        // perform bluetooth connections
        mChatService = new BluetoothChatService(this, mHandler);

        // Initialize the buffer for outgoing messages
        mOutStringBuffer = new StringBuffer("");
    }

    @Override
    public synchronized void onPause() {
        super.onPause();
        if(D) Log.e(TAG, "- ON PAUSE -");
    }

    @Override
    public void onStop() {
        super.onStop();
        if(D) Log.e(TAG, "-- ON STOP --");
```

```
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Stop the Bluetooth chat services
        if (mChatService != null) mChatService.stop();
        if(D) Log.e(TAG, "--- ON DESTROY ---");
    }

    private void ensureDiscoverable() {
        if(D) Log.d(TAG, "ensure discoverable");
        if (mBluetoothAdapter.getScanMode() !=
            BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
            Intent discoverableIntent = new Intent(
                BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
            discoverableIntent.putExtra(BluetoothAdapter.
                EXTRA_DISCOVERABLE_DURATION, 300);
            startActivity(discoverableIntent);
        }
    }

    /**
     * Sends a message.
     * @param message  A string of text to send.
     */
    private void sendMessage(String message) {
        // Check that we're actually connected before
        // trying anything
        if (mChatService.getState() != BluetoothChatService.
                STATE_CONNECTED) {
            Toast.makeText(this, R.string.not_connected, Toast.
                LENGTH_SHORT).show();
            return;
        }

        // Check that there's actually something to send
        if (message.length() > 0) {
            // Get the message bytes and tell the
            // BluetoothChatService to write
            byte[] send = message.getBytes();
            mChatService.write(send);
```

```
            // Reset out string buffer to zero and clear the
            // edit text field
            mOutStringBuffer.setLength(0);
            mOutEditText.setText(mOutStringBuffer);
        }
    }

    // The action listener for the EditText widget, to
    // listen for the return key
    private TextView.OnEditorActionListener mWriteListener =
        new TextView.OnEditorActionListener() {
        public boolean onEditorAction(TextView view, int actionId,
                KeyEvent event) {
            // If the action is a key-up event on the return key,
            // send the message
            if (actionId == EditorInfo.IME_NULL && event.getAction()
                 == KeyEvent.ACTION_UP) {
                String message = view.getText().toString();
                sendMessage(message);
            }
            if(D) Log.i(TAG, "END onEditorAction");
            return true;
        }
    };

    // The Handler that gets information back from the
    // BluetoothChatService
    private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case MESSAGE_STATE_CHANGE:
                if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " +
                        msg.arg1);
                switch (msg.arg1) {
                case BluetoothChatService.STATE_CONNECTED:
                    mTitle.setText(R.string.title_connected_to);
                    mTitle.append(mConnectedDeviceName);
                    mConversationArrayAdapter.clear();
                    BluetoothChat.this.sendMessage("Jerks: " +
                        jerkPointArray_[0] + " " +
```

```
                    jerkPointArray_[1] + " " +
                    jerkPointArray_[2] + " " +
                   jerkPointArray_[3] + " " +
                   jerkPointArray_[4] + " " +
                    "Times: " +
                   timeDifferenceArray_[0] + " " +
                   timeDifferenceArray_[1] + " " +
                   timeDifferenceArray_[2] + " " +
                   timeDifferenceArray_[3]);
               break;
           case BluetoothChatService.STATE_CONNECTING:
               mTitle.setText(R.string.title_connecting);
               break;
           case BluetoothChatService.STATE_LISTEN:
           case BluetoothChatService.STATE_NONE:
               mTitle.setText(R.string.title_not_connected);
               break;
           }
           break;
       case MESSAGE_WRITE:
           byte[] writeBuf = (byte[]) msg.obj;
           // construct a string from the buffer
           String writeMessage = new String(writeBuf);
           break;
       case MESSAGE_READ:
           byte[] readBuf = (byte[]) msg.obj;
           // construct a string from the valid bytes
           // in the buffer
           String readMessage = new String(readBuf,
                   0, msg.arg1);
           mConversationArrayAdapter.add(
                   mConnectedDeviceName+":  " + readMessage);
           theMessage_ = readMessage;
           gestureCheckHandler_.postDelayed(
                   gestureCheckRunnable_, 1000);
           break;
       case MESSAGE_DEVICE_NAME:
           // save the connected device's name
           mConnectedDeviceName = msg.getData().
                   getString(DEVICE_NAME);
           Toast.makeText(getApplicationContext(), "Connected to "
                   + mConnectedDeviceName,
```

```
                    Toast.LENGTH_SHORT).show();
            break;
        case MESSAGE_TOAST:
            Toast.makeText(getApplicationContext(),
                    msg.getData().getString(TOAST),
                    Toast.LENGTH_SHORT).show();
            break;
        }
    }
};

public void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    if(D) Log.d(TAG, "onActivityResult " + resultCode);
    switch (requestCode) {
    case REQUEST_CONNECT_DEVICE:
        // When DeviceListActivity returns with a device
        // to connect
        if (resultCode == Activity.RESULT_OK) {
            // Get the device MAC address
            String address = data.getExtras()
                    .getString(
                    DeviceListActivity.EXTRA_DEVICE_ADDRESS);
            // Get the BLuetoothDevice object
            BluetoothDevice device = mBluetoothAdapter.
                    getRemoteDevice(address);
            // Attempt to connect to the device
            mChatService.connect(device);
        }
        break;
    case REQUEST_ENABLE_BT:
        // When the request to enable Bluetooth returns
        if (resultCode == Activity.RESULT_OK) {
            // Bluetooth is now enabled, so set up a chat session
            setupChat();
        } else {
            // User did not enable Bluetooth or an error occured
            Log.d(TAG, "BT not enabled");
            Toast.makeText(this, R.string.bt_not_enabled_leaving,
                    Toast.LENGTH_SHORT).show();
            finish();
        }
```

```
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.option_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.scan:
            // Launch the DeviceListActivity to see devices
            // and do scan
            Intent serverIntent = new Intent(this,
                DeviceListActivity.class);
            startActivityForResult(serverIntent,
                REQUEST_CONNECT_DEVICE);
            return true;
        case R.id.discoverable:
            // Ensure this device is discoverable by others
            ensureDiscoverable();
            return true;
        }
        return false;
    }
}
```

14.2 Jerky: res/layout

14.2.1 Jerky: res/layout/bluemain.xml

The Jerky bluemain.xml file is the same as that in Appendix A.

14.2.2 Jerky: res/layout/customtitle.xml

The Jerky customtitle.xml file is the same as that in Appendix A.

14.2.3 Jerky: res/layout/devicelist.xml

The Jerky devicelist.xml file is the same as that in Appendix A.

14.2.4 Jerky: res/layout/devicename.xml

The jerky devicename.xml file is the same as that in Appendix A.

14.2.5 Jerky: res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.
    android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <ImageView android:layout_height="wrap_content" android:
        src="@drawable/jerky" android:id="@+id/imageView1"
        android:layout_width="wrap_content"></ImageView>
    <Button android:layout_gravity="center_horizontal"
                android:text="Reset"
                android:id="@+id/resetButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
    </Button>
    <LinearLayout android:layout_height="wrap_content"
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent">
        <TextView android:layout_weight="1"
        android:layout_height="wrap_content"
        android:textAppearance=
        "?android:attr/textAppearanceMedium"
```

```
        android:text="Jerk count: "
        android:id="@+id/textView1"
         android:layout_width="wrap_content">
        </TextView>
        <TextView android:layout_weight="1"
        android:layout_height="wrap_content"
        android:textAppearance=
        "?android:attr/textAppearanceMedium"
        android:text="0" android:id=
        "@+id/jerkCountTextView"
        android:layout_width="wrap_content">
        </TextView>
    </LinearLayout>
</LinearLayout>
```

14.2.6 Jerky: res/layout/message.xml

The Jerky message.xml file is the same as that in Appendix A.

14.2.7 Jerky: res/menu/optionmenu.xml

The Jerky message.xml file is the same as that in Appendix A.

14.2.8 Jerky: res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Jerky</string>

    <!-- BluetoothChat -->
    <string name="send">Send</string>
    <string name="not_connected">You are not connected
        to a device</string>
    <string name="bt_not_enabled_leaving">Bluetooth was not
        enabled. Leaving Bluetooth Chat.</string>
    <string name="title_connecting">connecting...</string>
    <string name="title_connected_to">connected: </string>
```

```
    <string name="title_not_connected">not connected</string>

    <!-- DeviceListActivity -->
    <string name="scanning">scanning for devices...</string>
    <string name="select_device">select a device to connect</string>
    <string name="none_paired">No devices have been paired</string>
    <string name="none_found">No devices found</string>
    <string name="title_paired_devices">Paired Devices</string>
    <string name="title_other_devices">Other Available
        Devices</string>
    <string name="button_scan">Scan for devices</string>

    <!-- Options Menu -->
    <string name="connect">Connect a device</string>
    <string name="discoverable">Make discoverable</string>
</resources>
```

14.2.9 Jerky: res/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.
        com/apk/res/android"
      package="com.org.magnum"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9" />
    <uses-permission android:name="android.permission.
        BLUETOOTH"></uses-permission>
    <uses-permission android:name="android.permission.
        BLUETOOTH_ADMIN"></uses-permission>

    <application android:icon="@drawable/icon" android:
        label="@string/app_name">
        <activity android:name=".JerkyActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.
                        LAUNCHER" />
            </intent-filter>
```

```
        </activity>
                <activity android:name=".BluetoothChat"
                  android:label="@string/app_name"
                  android:configChanges="orientation|keyboardHidden"
        />
        <activity android:name=".DeviceListActivity"
                  android:label="@string/select_device"
                  android:theme="@android:style/Theme.Dialog"
                  android:configChanges="orientation|keyboardHidden"
        />
    </application>
</manifest>
```

Chapter 15

Appendix C

Appendix C provides the code used to construct the lookup table for PENDING. Figure 15.1 is the file structure in Eclipse necessary for rebuilding the Android application used for constructing a look-up table.

15.1 LUT: src

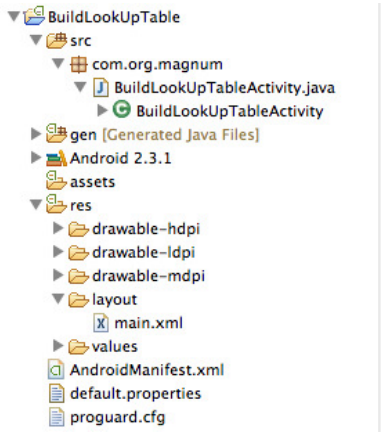
15.1.1 LUT: src/BuildLookUpTable.java

```
package com.org.magnum;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import android.app.Activity;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
```

Figure 15.1: Lookup table file structure.




```
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class BuildLookUpTableActivity extends Activity
        implements OnClickListener, LocationListener{
    TextView longitudeTextView_, latitudeTextView_;
    Button logButton_;
    EditText locationEditText_;
    String location_;

    //GPS elements
    LocationManager locationManager_;
    double latitude_, longitude_;

    //file writing elements
        File root_, file_;
        FileWriter fileWriter_;

        /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        locationManager_ = (LocationManager)
                getSystemService(LOCATION_SERVICE);
```

```
latitude_ = 0.00;
longitude_= 0.00;

longitudeTextView_ = (TextView) findViewById(
        R.id.longitudeTextView);
latitudeTextView_ = (TextView) findViewById(
        R.id.latitudeTextView);

logButton_ = (Button) findViewById(R.id.logButton);
logButton_.setOnClickListener(this);

locationEditText_ = (EditText) findViewById(
        R.id.locationEditText);

//get the SD card location
root_ = Environment.getExternalStorageDirectory();

try{
        file_ = new File(root_, "LUT.txt");
        if(!file_.exists())
        {
                file_.createNewFile();
        }
}
catch (IOException e) {
    //You'll need to add proper error handling here
        Toast.makeText(this, "File creation fail",
                Toast.LENGTH_SHORT).show();
}

try {
                fileWriter_ = new FileWriter(file_);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                Toast.makeText(this, "Filewriter
                        creation fail",
                        Toast.LENGTH_SHORT).show();
        }

locationManager_.requestLocationUpdates(LocationManager.
        GPS_PROVIDER, 15000, 10, this);
```

```
    }

        @Override
        public void onClick(View arg0) {
                // TODO Auto-generated method stub
                longitudeTextView_.setText(Double.toString(
                        longitude_));
                latitudeTextView_.setText(Double.toString(
                        latitude_));
                location_ = locationEditText_.getText().toString();
                try {
                        fileWriter_.append(location_ + ": " +
                                 Double.toString(latitude_) + " " +
                                Double.toString(longitude_) +
                                "\r\n");
                } catch (IOException e) {
                        e.printStackTrace();
                        Toast.makeText(getApplicationContext(),
                                "File write fail", Toast.
                                LENGTH_SHORT).show();
                }
                try {
                        fileWriter_.flush();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        Toast.makeText(getApplicationContext(),
                                "Flush fail", Toast.
                                LENGTH_SHORT).show();
                }
        }

        @Override
        public void onLocationChanged(Location arg0) {
                // TODO Auto-generated method stub
                longitude_ = arg0.getLongitude();
                latitude_ = arg0.getLatitude();
        }

        @Override
        public void onProviderDisabled(String arg0) {
                // TODO Auto-generated method stub
```

```
        }

        @Override
        public void onProviderEnabled(String arg0) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onStatusChanged(String arg0, int arg1,
                Bundle arg2) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onPause(){
                super.onPause();
                locationManager_.removeUpdates(this);
        }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        locationManager_.removeUpdates(this);
        try {
                fileWriter_.close();
        }
        catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
    }
}
```

15.2 LUT: res/layout

15.2.1 LUT: res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.
        android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView android:layout_gravity="center_horizontal"
                  android:text="Build Look-up Table"
                  android:layout_height="wrap_content"
                  android:id="@+id/textView1"
                  android:textAppearance="?android:attr/
                        textAppearanceMedium"
                  android:layout_width="wrap_content">
    </TextView>
    <Button android:layout_gravity="center_horizontal"
                android:text="Log"
                android:id="@+id/logButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
    </Button>
    <EditText android:layout_gravity="center_horizontal"
                  android:layout_height="wrap_content"
                  android:id="@+id/locationEditText"
                  android:layout_width="match_parent">
        <requestFocus></requestFocus>
    </EditText>
    <LinearLayout android:layout_width="match_parent" android:
        layout_height="wrap_content" android:id="@+id/linearLayout1">
        <TextView android:text="Latitude: "
                          android:layout_height="wrap_content"
                          android:id="@+id/textView2"
                          android:textAppearance="?android:
                                attr/textAppearanceMedium"
                          android:layout_width="wrap_content">
                                </TextView>
        <TextView android:text=""
                          android:layout_height="wrap_content"
                          android:id="@+id/latitudeTextView"
                          android:textAppearance="?android:attr/
                                textAppearanceMedium"
```

```
                        android:layout_width="wrap_content">
        </TextView>
    </LinearLayout>
    <LinearLayout android:layout_width="match_parent" android:
        layout_height="wrap_content" android:id="@+id/linearLayout2">
        <TextView android:text="Longitude: "
                        android:layout_height="wrap_content"
                        android:id="@+id/textView3"
                        android:textAppearance="?android:
                                attr/textAppearanceMedium"
                        android:layout_width="wrap_content">
        </TextView>
        <TextView android:text=""
                        android:layout_height="wrap_content"
                        android:id="@+id/longitudeTextView"
                        android:textAppearance="?android:
                                attr/textAppearanceMedium"
                        android:layout_width="wrap_content">
        </TextView>
    </LinearLayout>
</LinearLayout>
```

15.2.2 LUT: res/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.org.magnum"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9" />
    <uses-permission android:name="android.permission.
        ACCESS_COARSE_LOCATION"></uses-permission>
    <uses-permission android:name="android.permission.
        ACCESS_FINE_LOCATION"></uses-permission>

    <application android:icon="@drawable/icon" android:label=
        "@string/app_name">
        <activity android:name=".BuildLookUpTableActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

```
                <category android:name="android.intent.category.
                        LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Chapter 16

Appendix D

Appendix D provides all the code needed to build the PENDING Android application. Figure **??** is an image of the file structure in Eclipse necessary to build PENDING. The sections and subsections of this appendix form the outline for the folder and file names, respectively.

16.1 PENDING: src

16.1.1 PENDING: src/CalendarActivity.java

```
package com.org.magnum;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
```

```
public class CalendarActivity extends Activity{
        String filename_, currentLine_;
        ListView calendarListView_;
        ArrayAdapter<String> appointmentArrayAdapter_;
        InputStream inputStream_;
        InputStreamReader inputStreamReader_;
        BufferedReader bufferedReader_;

        /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        calendarListView_ = (ListView) findViewById(R.id.
                calendarListView);

        filename_ = Environment.getExternalStorageDirectory().
                getAbsolutePath().toString() +
                File.separator.toString() +
                "appointments.txt";

        appointmentArrayAdapter_ = new ArrayAdapter
                <String>(this, R.layout.list_item);

        calendarListView_.setAdapter(appointmentArrayAdapter_);

        try {
                inputStream_ = openFileInput(filename_);

                if(inputStream_ != null){
                        inputStreamReader_ = new InputStreamReader(
                                inputStream_);
                        bufferedReader_ = new BufferedReader(
                                inputStreamReader_);

                        while((currentLine_ = bufferedReader_.
                                readLine()) != null){
                                appointmentArrayAdapter_.
                                add(currentLine_);
                        }
```

```
                }

                inputStream_.close();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        Toast.makeText(this, "Filewriter
                                creation fail", Toast.
                        LENGTH_SHORT).show();
                }
    }
}
```

16.1.2 PENDING: src/ScheduleAppointmentActivity.java

```
package com.org.magnum;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Calendar;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.TimePicker;
import android.widget.Toast;

public class ScheduleAppointmentActivity extends Activity
        implements OnClickListener{
        Button confirmButton_;
        EditText destinationEditText_, contactEditText_;
        String destinationString_, contactString_;
        Integer day_, month_, year_, hour_, minute_,
                currentDay_, currentMonth_, currentYear_,
                currentHour_, currentMinute_;
```

```
    TimePicker timePicker_;
    DatePicker datePicker_;
    Calendar currentTime_;
    boolean validTimeandDate_;
    File root_, file_;
    FileWriter fileWriter_;

    @Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.appointment);

    confirmButton_ = (Button) findViewById(R.id.confirmButton);
    confirmButton_.setOnClickListener(this);

    destinationEditText_ = (EditText) findViewById(
            R.id.destinationEditText);
    contactEditText_ = (EditText) findViewById(
            R.id.contactEditText);

    timePicker_ = (TimePicker) findViewById(R.id.timePicker);
    datePicker_ = (DatePicker) findViewById(R.id.datePicker);

    validTimeandDate_ = false;

    root_ = Environment.getExternalStorageDirectory();

    try{
            file_ = new File(root_, "appointments.txt");
            if(!file_.exists())
            {
                    file_.createNewFile();
            }
    }
    catch (IOException e) {
        //You'll need to add proper error handling here
            Toast.makeText(this, "File creation fail",
                    Toast.LENGTH_SHORT).show();
    }

    try {
                    fileWriter_ = new FileWriter(file_);
```

```
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        Toast.makeText(this, "Filewriter
                                creation fail", Toast.
                                LENGTH_SHORT).show();
                }
        }

        @Override
        public void onClick(View arg0) {
                // TODO Auto-generated method stub
                switch(arg0.getId()){
                        case R.id.confirmButton:
                                destinationString_ =
                                        destinationEditText_.
                                        getText().toString();
                                contactString_ = contactEditText_.
                                        getText().toString();
                                day_ = datePicker_.getDayOfMonth();
                                month_ = datePicker_.getMonth();
                                year_ = datePicker_.getYear();
                                hour_ = timePicker_.getCurrentHour();
                                minute_ = timePicker_.
                                        getCurrentMinute();
                                currentTime_ = Calendar.getInstance();
                                currentMinute_ = currentTime_.get(
                                        Calendar.MINUTE);
                                currentHour_ = currentTime_.get(
                                        Calendar.HOUR);
                                currentDay_ = currentTime_.get(
                                        Calendar.DATE);
                                currentMonth_ = currentTime_.get(
                                        Calendar.MONTH);
                                currentYear_ = currentTime_.get(
                                        Calendar.YEAR);

                                if((destinationString_ != "") &&
                                        (contactString_ != "")){
                                        if(currentYear_ == year_){
                                        if(currentMonth_ == month_){
                                        if(currentDay_ == day_){
```

```
                                        if(currentHour_ < hour_){
                                        validTimeandDate_ = true;
                                        }
                                        else if(currentHour_ ==
                                        hour_){
                                        if(currentMinute_ < minute_){
                                        validTimeandDate_ = true;
                                        }
                                        }
                                        else{
                                        validTimeandDate_ = false;
                                        }
                                        }
                                        else if(currentDay_ > day_){
                                        validTimeandDate_ = false;
                                        }
                                        else{
                                        validTimeandDate_ = true;
                                        }
                                        }
                                        else if(currentMonth_ >
                                        month_){
                                        validTimeandDate_ = false;
                                        }
                                        else{
                                        validTimeandDate_ = true;
                                        }
                                        }
                                        else if(currentYear_ > year_){
                                        validTimeandDate_ = false;
                                        }
                                        else{
                                        validTimeandDate_ = true;
                                        }
                                }
                        break;
                }
                if(validTimeandDate_){
                        try {
                                fileWriter_.append(
                                destinationString_ + "," +
                                contactString_ + "," +
```

```
                                month_ + "," + day_ +
                                "," + year_ + "," + hour_
                                + "," + minute_);
                        } catch (IOException e) {
                                e.printStackTrace();
                                Toast.makeText(
                                getApplicationContext(),
                                "File write fail",
                                Toast.LENGTH_SHORT).show();
                        }
                        try {
                                fileWriter_.flush();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                                Toast.makeText(
                                getApplicationContext(),
                                "Flush fail", Toast.
                                LENGTH_SHORT).show();
                        }
                        try {
                                fileWriter_.close();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }
}
```

16.2 PENDING: res/layout

16.2.1 PENDING: res/layout/calendar.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
```

```
    <ListView android:layout_height="1000px"
                  android:id="@+id/calendarListView"
                  android:layout_width="match_parent">
        </ListView>
        <Button android:layout_height="wrap_content"
                        android:layout_width="wrap_content"
                        android:id="@+id/initiateButton"
                        android:text="Initiate">
        </Button>

</LinearLayout>
```

16.2.2 PENDING: res/layout/appointment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <TextView android:layout_gravity="center_horizontal"
                  android:layout_height="wrap_content"
                  android:textAppearance="?android:
                        attr/textAppearanceMedium"
                  android:layout_width="wrap_content"
                  android:id="@+id/textView1"
                  android:text="Schedule new appointment:">
    </TextView>
    <LinearLayout android:layout_height="wrap_content"
        android:layout_width="match_parent" android:id=
        "@+id/linearLayout1">
        <TextView android:layout_height="wrap_content"
                        android:textAppearance="?android:
                        attr/textAppearanceMedium"
                        android:layout_width="wrap_content"
                        android:id="@+id/textView2"
                        android:text="Destination: ">
        </TextView>
        <EditText android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:layout_width="wrap_content"
```

```
                        android:id="@+id/destinationEditText">
            <requestFocus></requestFocus>
        </EditText>
    </LinearLayout>
    <LinearLayout android:layout_height="wrap_content"
        android:layout_width="match_parent" android:id=
        "@+id/linearLayout2">
        <TextView android:layout_height="wrap_content"
                        android:textAppearance="?android:
                        attr/textAppearanceMedium"
                        android:layout_width="wrap_content"
                        android:id="@+id/textView3"
                        android:text="Contact: "></TextView>
        <EditText android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:layout_width="wrap_content"
                        android:id="@+id/contactEditText">
        </EditText>
    </LinearLayout>
    <DatePicker android:id="@+id/datePicker"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content">
    </DatePicker>
    <TimePicker android:id="@+id/timePicker"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content">
    </TimePicker>
    <Button android:layout_gravity="center_horizontal"
                android:text="Confirm"
                android:id="@+id/confirmButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
    </Button>

</LinearLayout>
```

Bibliography

[1] K. Clark, S. Peters, and M. Tomlinson, "The determinants of lateness: evidence from british workers," *Scottish Journal of Political Economy*, vol. 52, no. 2, pp. 282–304, May 2005.

[2] G. Blau, "Influence of group lateness on individual lateness: a cross-level examination," *The Acadamy of Management Journal*, vol. 38, no. 5, pp. 1483–1496, Oct. 1995.

[3] W. Wolf, "Cyber-physical systems," *Computer*, vol. 42, no. 3, pp. 88–89, Mar. 2010.

[4] R. Rajkumar, I. Lee, and L. Sha, "Cyber-physical systems - the next computing revolution," *Proceedings of Design and Automation Conference*, 2010.

[5] J. Shi, J. Wan, and H. Yan, "A suvery of cyber-physical systems," in *Proceedings of the International Conference on Wireless Communications and Signal Processings*, 2011.

[6] N. Ohmori, M. Nakazato, and N. Harata, "Gps mobile phone-based activity diary survey," in *Proceedings of the Eastern Asia Society for Transportation Studies*, vol. 5, 2005, pp. 1104–1115.

[7] Y. Lui and S. Tian, "Research of indoor gps signals acquisition algorithm," in *4th International Conference on Wireless Communications, Networking and Mobile Computing*, 2008, pp. 1–4.

[8] K. Cheung, S. Intille, and K. Larson, "An inexpensive bluetooth-based indoor positioning hack," *Proceedings of UbiComp'06 Extended Abstracts*, 2006.

[9] J. Rekimoto, T. Miyaki, and T. Ishizawa, "Lifetag: Wifi-based continuous location logging for life pattern analysis," in *Proceedings of the 3rd International Symposium on Location-and Context-Awareness*, vol. 4718. Springer-Verlag New York Inc, 2007, p. 35.

[10] G. Dedes and A. Dempster, "Indoor gps positioning-challenges and opportunities," in *Proceedings of the 62nd IEEE Vehicular Technology Conference*, vol. 1. IEEE, 2005, pp. 412–415.

[11] F. van Diggelen, "Indoor gps theory & implementation," in *IEEE Position Location and Navigation Symposium.* IEEE, 2002, pp. 240–247.

[12] Y. Nakazato, M. Kanbara, and N. Yokoya, "Localization system for large indoor environments using invisible markers," in *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology.* ACM, 2008, pp. 295–296.

[13] A. Kobelev, I. Sergeez, and S. Schookin, "Smartphone-based mobile solutions for health control in humans."

[14] J. Vainio, "Bluetooth security," Seminar on Internetworking: Ad Hoc Networking, 2000.

[15] L. Wong, "Potential bluetooth vulnerabilities in smartphones," 2005.

[16] B. Potter, "Bluetooth vulnerabilities," *Network Security*, no. 3, pp. 4–5, Mar. 2004.

[17] C. Hager and S. Midkiff, "An analysis of bluetooth security vulnerabilities," *Wireless Communications and Networking, IEEE*, vol. 3, pp. 1825–1831, Mar. 2003.

[18] ——, "Demonstrating vulnerabilities in bluetooth security," in *Proceedings of the Annual IEEE Global Telecommunications Conference*, vol. 3, Dec. 2003, pp. 1420–1424.

[19] A. Solon, M. Callaghan, J. Harkin, and T. McGinnity, "Case study on the bluetooth vulnerabilities of mobile devices," *International Journal of Computer Science and Network Security*, vol. 6, no. 4, pp. 125–129, Apr. 2006.

[20] P. Asare, D. Cong, and B. Kim, "The medical device dongle: an open-source standards-based platform for interoperable medical device connectivitiy," in *Proceedings of the 2nd ACM International Health Informatics Symposium.* ACM, Jan. 2012, pp. 667–671.

[21] M. Patel and J. Wang, "Applicaitons, challenges, and prospective in emerging body area networking technologies," *IEEE Wireless Communciations*, pp. 80–88, Feb. 2010.

[22] U. Lee, S. Jung, and D. Cho, "P2p content distribution to mobile bluetooth users," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 1, pp. 356–367, Jan. 2010.

[23] H. Hossain, U. Kabir, and S. Rahman, "Modified approach of rfcomm implementation to protect bluetooth technology from bluebug attack," *IJCIT*, vol. 1, no. 2, pp. 20–23, 2011.

[24] T. Georgiev and E. Georgieva, "Mobile application for determination of users' text entry speed," *Proceedings of the International Conference on Computer Systems and Technologies*, 2008.

[25] J. de Querioz, J. Fechine, and A. Barbosa, "Towards a multideimensional approach for the evaluation of multimodal application user interfaces," *Human-Computer Interaction*, pp. 29–38, 2009.

[26] J. Dunning, "Taming the blue beast," *IEEE Mobile Device Security*, pp. 20–27, Mar. 2010.

[27] J. Liu, L. Zhong, and J. Wickramasuriya, "uwave: accelerometer-based personalized gesture recongition and its applications," *Pervasive and Mobile Computing*, vol. 5, no. 657-675, July 2009.

[28] J. Kim, J. He, K. Lyons, and T. Starner, "The gesture watch: a wireless contact-free gesture based wrist interface," *11th IEEE International Symposium on Wearable Computers*, pp. 15–22, Oct. 2007.

[29] L. Tarrataca, A. Santos, and J. Cardoso, "The current feasibility of gesture recognition for a smartphone using j2me," in *Proceedings of the ACM Symposium on Applied Computing*. ACM, 2009, pp. 1642–1649.

[30] K. Oorschot, J. Bertrand, and C. Rutte, "Field studies into the dynamics of product development tasks," pp. 720–739, 2005.

[31] J. Wagner, "Maximizing profits in a tight market through optimization."

[32] M. Bassett and R. Engineers, "Client time expectations and construction industry performance."

[33] N. Clarke and S. Furnell, "Authentication of users on combile telephones - a survey of attitudes and practices," *Computers and Security*, vol. 24, no. 7, pp. 519–527, 2005.

[34] J. Phillips, E. Charles, T. Ting, and S. Demurjian, "Information sharing and security in dynamic coalitions," in *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*. ACM, 2002, pp. 87–96.

[35] O. S. Industries, "Optical sensors in smart mobile devices," 2010.

[36] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the Annual USENIX Conference*. USENIX Association, 2010, pp. 21–21.

[37] R. Hansen, R. Wind, C. Jensen, and B. Thomsen, "Seamless indoor/outdoor positioning handover for location-based services in streamspin," in *The 10th International Conference on Mobile Data Management: Systems, Services and Middleware*. IEEE, 2009, pp. 267–272.

[38] M. Koslowsky, "A new perspective on employee lateness," *Applied Psychology: An International Review*, vol. 49, no. 3, pp. 390–407, 2000.

[39] J. Elicker, M. Foust, A. O'Malley, and P. Levy, "Employee lateness behavior: the role of lateness climate and individual lateness attitude," in *Human Performance*, vol. 21, no. 4, 2008, pp. 421–441.

[40] J. Gotow, L. Zienkiewicz, J. White, and D. Schmidt, "Addressing challenges with augmented reality applications on smartphones," in *Mobile Wireless Middleware, Operating Systems, and Applications*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2010, vol. 48, pp. 129–143.

[41] A. Santos, J. Cardoso, D. Ferreira, and P. Diniz, "Mobile context provider for social networking," in *On the Move to Meaningful Internet Systems Workshops*, ser. Lecture Notes in Computer Science, T. D. R. Meersman, P. Herrero, Ed. Springer Berlin Heidelberg, 2009, vol. 5872, pp. 464–473.

[42] K. Kawauchil, T. Miyaki, and J. Rekimoto, "Directional beaconing: a robust wifi positioning method using angle-of-emission information," in *Proceedings of the 4th International Symposium on Location and Context Awareness*, 2009, pp. 103–119.

[43] M. Bargh and R. Groote, "Indoor localization based on response rate of bluetooth inquiries," in *Proceedings of the 1st ACM International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments*. ACM, 2008, pp. 49–54.

[44] B. Fischer, G .and Dietrich and F. Winkler, "Bluetooth indoor localization system," in *Proceedings of the 1st Workshop on Positioning, Navigation and Communication*, 2004, pp. 147–156.

[45] C. Hsu and C. Yu, "An accelerometer based approach for indoor localization," in *Proceedings of the Symposium on Ubiquitous, Autonomic and Trusted Computing*, 2009, pp. 223–227.

[46] H. Shin, J. Chon, and H. Cha, "Smartslam: constructing an indoor floor plan using smartphones," 2010.

[47] A. Ofstad, E. Nicholas, and R. Szcodronski, "Aampl: accelerometer augmented mobile phone localization," in *Proceedings of the 1st ACM International Workshop on Mobile Entry Localization and Tracking in GPS-less Environments*, 2008, pp. 13–18.

[48] S. Thrun, W. Burgard, and D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Autonomous Robots*, vol. 5, pp. 253–271, 1998.

[49] F. Dellaert, D. Fox, W. Burgard, and W. Thrun, "Monte carlo localization for mobile robots," in *Robotics and Automation*, vol. 2, 1999, pp. 1322–1328.

[50] B. Fehmi, A. Phillips, and T. Henderson, "Less is more: energy-efficient mobile sensing with senseless," in *Proceedings of the 1st ACM workshop on Networking, Systems, and Applications for Mobile Handhelds*. ACM, 2009, pp. 61–62.

[51] J. Chon and C. Hojung, "Lifemap: a smartphone-based context provider for location-based services," *Pervasive Computing, IEEE*, vol. 10, no. 2, pp. 58–67, Feb. 2011.

[52] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: ACM, 2010, pp. 299–314.

[53] H. Mahmoud, J. Gallegos, and D. Vucci, "Adaptive gps duty cycling."

[54] Y. Chon, E. Talipov, and H. Shin, "Mobility prediction-based smartphone energy optimization for everyday location monitoring," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011, pp. 82–95.

[55] J. Kang, D. Kim, E. Kim, Y. Kim, S. Yoo, and D. Wi, "Seamless mobile robot localization service framework for integrated localization systems," *Wireless Pervasive Computing*, pp. 175–179, 2008.

[56] M. Klepal, M. Weyn, W. Najib, I. Bylemans, S. Wibowo, W. Widyawan, and B. Hantono, "Ols: opportunistic localization system for smart phones devices," in *Proceedings of the 1st ACM workshop on Networking, Systems, and Applications for Mobile Handhelds*. ACM, 2009, pp. 79–80.

[57] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson, "Easytracker: automatic transit tracking, mapping, and arrival time prediction using smartphones," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2011, pp. 68–81.

[58] M. Joselli and R. Clua, "grmobile: a framework for touch and accelerometer gesture recognition for mobile games," in *The 7th Brazilian Symposium on Games and Digital Entertainment*, 2009, pp. 141–150.