University of Cape Town

Department of Computer Science

Honours Project Report

Server-Based Device Discovery for Android

Author
Bryan Davies

Supervisor
Prof. Gary Marsden

	Category	Min	Max	Chosen
1	Requirement Analysis and Design	0	20	**15**
2	Theoretical Analysis	0	25	**0**
3	Experiment Design and Execution	0	20	**17**
4	System Development and Implementation	0	15	**0**
5	Results, Findings and Conclusion	10	20	**18**
6	Aim Formulation and Background Work	10	15	**10**
7	Quality of Report Writing and Presentation	10		**10**
8	Adherence to Project Proposal and Quality of Deliverables	10		**10**
9	Overall General Project Evaluation	0	10	**0**
Total marks		**80**		**80**

29 October 2012

Abstract

Network and device discovery by mobile devices is common practice, and the methods developed to accomplish this task have been successfully utilized for many years. However, mobile devices generally a have a limited power supply, and these methods are not always the most efficient. This paper presents the device discovery system of GROUT: a mobile ad-hoc networking system. Herein, the main causes of battery usage related to device discovery are discussed, and alternative ways in which discovery can be performed are explored. A method that seeks to save power by minimizing the amount of time that power consuming radios spend scanning, and perform the discovery operations on a server rather than on the device itself, is proposed and implemented. This system is evaluated on its performance accuracy, its battery usage and the amount of data transferred during operation. The results of the evaluation show that while the system can work, it is heavily dependent on existing infrastructure, and Bluetooth devices that are within range. Battery usage results are inconclusive, and data transfer volume is reasonably low.

Acknowledgements

I would like to thank Professor Gary Marsden for his invaluable assistance, suggestions and all the time he dedicated to helping us complete this project.

I would like to thank Tsu-Shiuan Lin, and Sashen Singh for their support and hard work on this project.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Cellphones, tablets and other mobile devices are becoming more and more pervasive in our modern society. Fitchard (2012) reports that the average citizen of the United States of America owns 1.57 devices; 1.85 for the rest of the surveyed world. Some countries reportedly have nearly 280 mobile devices for every hundred citizens (CIA World factbook, 2012, as cited in Index mundi, date unknown). People use their mobile devices in nearly every facet of their lives; at school and at work, at parties, during exercise, and play. From phone calls to photographs, modern day devices share in a large part of our lives and they store a significant amount of our daily experiences. A recent survey revealed that social media disrupted the mobile games market; a market that was rivalling the pc and console games industry (Farago, 2012). Mobile device users clearly desire to share information with each other. Information sharing between mobile devices is not a new technology. The Short- and Multimedia Message Services (SMS and MMS respectively) have enabled cellphone users to share information across cellular networks. Infra-red and Bluetooth connections have enabled mobile device users to form a direct connection between their devices and share information in a way that does not use infrastructure networks at all. Wi-Fi radios are commonly installed on today's mobile devices, and yet systems that allow mobile devices to form ad-hoc networks with each other are still uncommon. GROUT will be a system that allows users to form a wireless connection between geographically co-located groups of devices, without requiring a connection to an infrastructure network.

1.2 Windows Phone 7

Microsoft's Windows Phone 7 (WP7) took a bold new approach when designing their user interface. They chose to adopt a different design metaphor than the popular app-based interface of Google's Android and Apple's iOS. The WP7 home screen (among other screens) displays a set of tiles, as can be seen in Figure 1.1. These tiles are the representation on the home screen of phone related objects; usually applications. They can also represent locally stored information such as photographs, calendar entries and contacts. Tiles can also represent web feeds, displaying certain information as it is updated on the internet. Tiles are also physically linked to those

objects; essentially a button, or icon that changes the focus of the device's display to the object. Most importantly to GROUT, a tile can represent the virtual manifestation of a contact, or person, on a Windows Phone 7 device.

Figure 1.1. Windows phone 7 tiles

1.3 GROUT

Grout is a substance that fills the gap between wall and floor tiles. The whole concept of GROUT revolves around the idea that people can be represented as Windows Phone 7 tiles. GROUT will be the application that "filled the gap" between individual tiles, so to speak. To put it more technically, GROUT will be a system that allows collocated devices to network with each other without needing a connection to existing network infrastructure. In the developed world, networking through infrastructure such as Wi-Fi access points, and cellular networks is common, and often affordable, if not free. Recently, however, many companies have begun doing away with their unlimited data plans (Yu, 2012). In countries such as South Africa, cellular data can be expensive (MTN, Date unknown). Very few systems exist that allow devices to connect to each other directly and for free. Those that do, such as

native Bluetooth connections, are not designed to handle the transfer of large volumes of data, whether they be in the form of files, or live streams. GROUT intends to allow these situations to occur. In order for GROUT to exist, three subsystems must be designed. Firstly, a subsystem that allows GROUT-able devices to identify each other, and the fact that they are collocated. This process is known as *device discovery* and is the focus of this report. A subsystem must exist that manages the connections between devices, and the data transfer between them. This subsystem deals with Mobile Ad-hoc Networking, and is the focus of a different report. The final subsystem would be the one that deals with the user interface for GROUT. This is what allows the user of a device interacts with the system itself, files and other data on their own device as well as data on other GROUT-able devices. This again is the focus of another report

1.4 Device discovery

This paper deals with the device discovery aspects of GROUT. Device discovery, in any form, must be able to perform two functions. Firstly, it must successfully become aware of other devices with which it can connect. A number of methods exist that allow a mobile device to become aware of their peers. Bluetooth and Wi-Fi are the most common non-cellular means through which mobile devices communicate. Both have a scanning function that allows the discovery of devices and access points, respectively. While the native scanning methods are functional and reliable, Bluetooth scanning is time consuming, and Wi-Fi scanning uses large quantities of battery power. There exist alternative methods that utilise the concepts of location, proximity and space in order to solve the device discovery problem. These often make use of either the Bluetooth, Wi-Fi or both radios. Such methods are reviewed in section 2: related work. Secondly, it must successfully gain information, unique to each of those devices that allow connections to be formed. In the case of Bluetooth networking, this information is the Bluetooth MAC address, in the case of Wi-Fi networking, this is the Wi-Fi MAC address. In order for the device discovery subsystem to interface properly with the Ad-hoc Networking subsystem, a list of Wi-Fi or Bluetooth MAC addresses of all the GROUT-able devices in the vicinity must be calculated. This list is the most important result of the device discovery's execution.

These two functions are usually completed by the device, and its Bluetooth and Wi-Fi radios alone; the process is self-contained and uses power only. But the alternative

mentioned above often require communication with a server, which uses data. As mentioned in section 1.3 above, data is not always cheap. If client-server communication is required, the amount of data transferred during communication should not be excessive.

1.4.1 Research questions

From the above description of the essentials of a device discovery system, the following research questions are asked.

1. Can a device discovery system be created that successfully harvests the MAC addresses of all collocated devices while saving power?

2. Can this device discovery system perform its function without transferring unreasonable amounts of data?

Question one is made of two factors; successfully collecting MAC addresses, and minimising the use of power. They are combined into one question because separately, solutions exist that answer these two questions. However, trying to combine the two in one system makes for interesting research. Thus the two factors are linked. Question two arises from the fact that in attempting to find answers to question one, data may need to be consumed; something not associated with the standard methods of performing device discovery. These questions will be kept in mind while developing the system, and the system will then be tested empirically on these three factors.

1.5 Outline

This remainder of this report begins with Chapter 2, which takes look at the reasons for the formulation of the research questions; Smartphone hardware and battery life. It looks at some of the research and work that inspired the idea behind the system that was developed, and the concepts that are associated with, and important to, the understanding of the system. In Chapter 3 it details the design of the system that was to be developed, followed by Chapter 4 which contains a description of the actual development process that went in to creating the GROUT device discovery system. Chapter 5 is a detailed evaluation of the final system. Finally, in Chapter 6, the results are critically analysed in terms of the research questions and the background work.

2. Background and related work

Being able to sense other devices or network infrastructure is critical to device discovery systems. This sections reviews some of the sensors most commonly used in device discovery, as well as means in which the can be used more efficiently

2.1 Battery life

Battery life is directly related to a device's utility (Pering, Agarwal, Gupta & Want, 2006) and is therefore an important factor in the user's experience of their device; thus power consumption is of concern to mobile researchers and developers. The power consumption of sensors commonly installed on mobile devices varies drastically, as can be seen from Table 2.1 (Abdesslem, Phillips & Henderson, 2009). However, these pieces of technology can be utilized more efficiently; either by themselves or together in ways that reduce the overall power consumption of the device.

Table 2.1

Taken from Abdesslem et al. (2009). The approximate battery life of the Nokia N95 8GB device while using sensor at full power and average power consumption of sensors over this period.

Sensor	*Approximate battery life (hrs)*	*Average power consumption (mW)*
Video camera	3.5	1258
IEEE 802.11	6.7	661
GPS(outdoors)	7.1	623
GPS (indoors)	11.6	383
Microphone	13.6	329
Bluetooth	21.0	211
Accelerometer	45.9	96
All sensors turned off	170.6	26

2.1.1 Wi-Fi and cellular radios

Wi-Fi and cellular radio are two common mediums through which mobile devices connect to non-cellular networks. 802.11 Wi-Fi is popular due in part to the large number of available of Wi-Fi hotspots in urban areas. The monetary expense associated with connecting through and transferring data across 3G and GSM networks is comparatively high to the expenses related to Wi-Fi network connections (Agarwal et al., 2007). Unfortunately, Wi-Fi is the medium that is most demanding on the device's battery. While the data transfer procedure for Wi-Fi is shown to be more efficient than 3G or GSM, the total power consumption for the Wi-Fi radio use is only less than the cellular radio when the data volume is sufficiently large (Agarwal et al., 2007; Balasubramanian, Balasubramanian & Venkatarami, 2009). Consumption has been shown to be due to the power overheads of the radio's scanning function while it looks for access points (APs) or other devices. (Abdesslem et al., 2009; Agarwal et al., 2007;Ananthanarayanan & Stoica, 2009; Balasubramanian et al., 2009; Rahmati & Zhong, 2007; Want & Pering, 2005). Agarwal and colleagues (2007) measured the scanning costs at around 70% of the total Wi-Fi power consumption. This is a large percentage of the power use. Given the average power usage seen in Table 2.1, Wi-Fi scanning is an action that should be used as little as possible. Thus GROUT device discovery will attempt to minimise the use of this action.

2.1.2 Bluetooth radios

Table 1.1 shows that Bluetooth uses significantly less power than Wi-Fi. Other studies have shown the Bluetooth is far more energy efficient at scanning for devices than any of the other radios on cellular phones (Pering, Raghunathan & Want, 2005). However, Bluetooth has some notable drawbacks that make it less than ideal for forming connections, and transferring data. Bluetooth is connection orientated, meaning that it needs to identify the device with which it will be communicating before any data can be transferred, a process that requires between ten seconds and half a minute to complete (Hal, Vawdrey & Knutson, 2002; Kinney, 2003). This is due to its complicated connection protocol as seen in Figure 2.1.

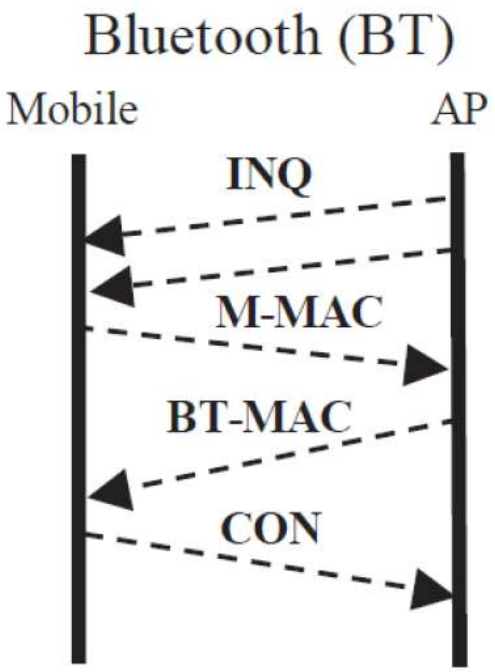

Figure 2.1. The Bluetooth connection protocol (Pering et al., 2005): Access Point sends stream of Inquiry packets; Mobile returns MAC address; AP returns its own MAC; Mobile creates connection at IP-level

Wi-Fi connections are established faster than Bluetooth connections (some 30 milliseconds as claimed by Kinney (2003)), because only the last two steps of the protocol in Figure 1 are executed; an AP broadcasts its MAC address periodically, and the mobile device utilises only this it to configure the connection.

The practical transfer rate of the Bluetooth radios commonly installed on current mobile systems is less than 2Mbps (Ananthanarayanan & Stoica, 2009; Pering et al., 2005). This is far slower than the 11Mbps that the Wi-Fi radios can attain on equivalent devices (Ananthanarayanan & Stoica, 2009; Pering et al., 2006; Pering et al., 2005)

Furthermore, the effective range of the Bluetooth radio is no more than 10m. Wi-Fi radios can form connection with other devices up to 100m away, depending on the situation and power of the AP. Interestingly, cellular radios can make a connection from a distance of up to 500m (Ananthanarayanan & Stoica, 2009).

For mobile devices, Wi-Fi would be the logical choice for forming a network connection aimed at sharing data (Pering et al., 2005). It is as fast as is necessary for the functions that the device supports. The range is suitable for urban and enterprise

environments wherein wireless networks are popular. The only drawback then is the power consumption. (Abdesslem et al., 2009; Ananthanarayanan & Stoica, 2009; Anastasi, Conti, Gregori & Passarella, 2008; Manweiler & Choudry, 2011; Pering et al., 2006; Pering et al., 2005; Sorber, Banerjeem Corner & Rollins, 2005; Want & Pering, 2005).

Given that Wi-Fi uses a lot of power, and Bluetooth uses less, but takes up more time, there is a trade-off between power and time can take place here, if both radios are used together. Wi-Fi specifically for its most efficient function; data transfer, and Bluetooth for its most efficient function; scanning. In addition, it is possible that the time detriments of the Bluetooth protocol are only a concern when the connection is only intended for one device. If many connections could be formed from a single scan, the benefits outweigh the detriments

The different distance associated with each radio are of interest too, as they can differentiate between sensed objects at different locations. In particular, Wi-Fi sensed objects are guaranteed to be within a certain distance of the sensor, but Cellular sensed objects might be close, or might be far way. A similar relationship exists between Bluetooth sensed objects and Wi-Fi sensed objects. This provides the sensor device with additional information that it can use to make decisions about the locality of the sensed devices.

2.2 Sensor efficiency

As we have established Wi-Fi is the optimal form of connection for data transfer. However, it is far from being the most power efficient sensor. By itself, the 802.11 sensor will drain a Smartphone's battery in few hours if it were to run continuously (Abdesslem et al., 2009). In many Smartphones, the Wi-Fi radio has some means that are designed to limit the amount of power that is used. One of the most common is the 802.11 standard known as Power Save Mode (PSM) (Anastasi et al., 2008; Balasubramanian et al., 2009; Manweiler & Choudry, 2011; Rahmati & Zhong, 2007). PSM saves power by alternately switching to sleep mode when there are no active data transfers. While this can reduce the amount of power consumed by a Wi-Fi from 661mW device to around 256mW (Pering et al., 2006), this mode can only be effective work when the device is actually connected to its network. The key word in the term "Mobile devices" is mobile; they tend move around with their users, and

therefore tend to move between different network access points. It stands to reason that they will need to discover these new network APs. While the radio is searching for another network, it will not enter the sleep mode of PSM (Anastasi et al., 2008). Thus, supplementory ways of reducing the amount of power used are needed.

2.2.1 Hierarchy of sensors

Using a combination of sensors to detect local networks is a method that has been explored by a number of researchers. The core idea behind this method is to switch dynamically between radios and interfaces of different power consumption (Ananthanarayanan & Stoica, 2009; Hall et al., 2002; Pering et al., 2006; Pering et al., 2005; Sorber et al., 2005). By switching to a low power sensor, one reduces the amount of power being used to a rate that suits the device's current context (Abdesslem et al., 2009; Pering et al., 2006; Sorber et al., 2005).

Pering and colleagues (Pering et al., 2005) made use of three wireless radio devices in increasing order of power consumption; the Mica2 Mote (Pering et al., 2005), Bluetooth, and 802.11b Wi-Fi. The system uses the low power radios to test perform periodic tests for possible networks in the vicinity, and only powers up the more costly radio to test if the network is legitimate. Additionally, the system can power down if the current transfer does not require the bandwidth of the stronger radio. Motes are not commonly installed on mobile devices; however, the other two radios are common and therefore of interest.

The connection protocol for Bluetooth → Wi-Fi process is almost identical to the one shown in Figure 1. The only difference is found in the final step, where the Wi-Fi MAC address of the AP is sent to the device instead of the Bluetooth Mac address. They show that by using a dual radio process, a significant amount of power is saved in the AP discovery process. When compared, traditional Wi-Fi discovery uses 6 times as much power. However, due to the time consuming Bluetooth connection protocol, discovery latency is almost tripled and connection latency grew by a factor of five. In total, the average latency grew from 1530 milliseconds to 4920 milliseconds. The conclusion reached by Pering et al. (2005) is that the increased latency is an acceptable loss considering the power savings attained, and is applicable due to the radios' continued inclusion in modern mobile devices.

Turducken is a system that is conceptually very similar to the one described in the section above only far more complicated. A Turducken is also made up of radios in hierarchical tiers. However, they are ranked on computational ability rather than network connection type. Turducken has a number of rules:

- Each tier has a number of operations that it must be able to execute.
- A tier must be able to perform a computational task.
- Ideally, the tier performing that task is the tier that performs it optimally.
- Each tier must, through its own devices or with messages from another tier, be made aware of external services that require its computation
- Each tier must be able to delegate service requests to the tier directly above or below itself.
- Lastly a tier must be able to put itself into a state of suspension.

Theoretically, a Turducken system could be composed of any number of hierarchical tiers. One advantage Turducken has over the other systems described in this review is that it could potentially save power through reduced cost of sensing, and reduced computational cost.

Ananthanarayanan & Stoica (2009) proposed Blue-Fi, which also exploits two non-Wi-Fi radios to assist in minimizing the power cost associated with constant Wi-Fi scanning. However, unlike Pering et al. (2005) and Turducken (Sorber et al., 2005), the cellular and Bluetooth radios are not involved in the actual connection process at all. Ananthanarayanan & Stoica (2009) recognize that humans are creatures of habit and as such are likely to encounter the same Cellular tower, Bluetooth devices and Wi-Fi networks repeatedly. Cellular Tower discovery is a radio that is not often disabled by choice (Ananthanarayanan & Stoica, 2009; Ramli & Hasbullah 2010). We have already established that Bluetooth is a low power radio, and therefore not disabled as often as Wi-Fi. When a Wi-Fi network is accessed for the first time, Blue-Fi records the identity of the nearest cellular network tower, and the MAC address of any Bluetooth devices in the vicinity. These are stored permanently on a server as that Wi-Fi network's context. The system then monitors the users, and registers when they re-enter a stored context. This allows Blue-Fi to predict when a user is within range of a Wi-Fi network again, and wakes the Wi-Fi radio so that a connection can be formed. The time consumption of the Bluetooth processes no

longer enter into the utility cost, as they are not at all associated with the connection process.

Current mobile devices are being released with a number of additional sensors and interfaces that allow the device to acquire local information, and become aware of their context. These interfaces include Video Cameras, Accelerometers, GPS and RFIDs in the form of Near Field Communication (NFC) components.

While not specifically orientated towards network discovery, Sensless (Abdesslem et al. 2009) uses the accelerometer sensor to gather basic contextual information. Essentially, Sensless makes use of the simple notion that if a device is not being moved, or has not been moved for a period of time, then it is not in use. For example, GPS is not needed by the user when they are sitting at their desk. The results from the user test performed by the researchers showed that battery usage was reduce by over 58% when their system was employed. Not only is this system easy to implement due to its use of a sensor common to most mobile devices, but its predicted power savings are comparable to those reported by Blue-Fi.

The goal of the systems addressed above is to use more sensors in different roles in order to save power. Persing et al. (2005) and Turducken did so by trying to combine the various sensor radios it utilised into one hybrid network AP sensor. This was shown to be successful, and thus is a legitimate strategy in the quest for saving power. Blue-Fi and Senseless utilise only the Wi-Fi radio to detect networks. The other sensors are used to gather information about the device's current context, in terms of location and motion respectively. To provide the device with an understanding of the specific space that currently surrounds it.

While the GROUT system most likely will not use the sensor hierarchy to save power, it will still attempt to minimize the time spent in the high powered states. Blue-Fi, in particular presents the idea that a device can be made aware of Wi-Fi access points without using the Wi-Fi radio at all. This idea is also very important to the GROUT discovery system.

2.3 Location

Blue-Fi is indicative of the fact that devices can become aware of the possible existence of Wi-Fi network access points without needing to perform a Wi-Fi scan. This is because the device is aware of its location in relation to the access point.

While location is familiar concept, Hightower & Borriello (2001) describe how we must be aware that location can be thought of in many different ways. An object's *physical position* can be thought of as being a quantitative measure of where it is situated. For example, a Global Positioning System can provide us with physical position of the object: 33.92S, 18.42 E and 15m above sea level. That same object can also have a number of *symbolic locations*; qualitative, abstract descriptions of the object's location that are justified with the inclusion of additional information. For example: *at school*; *in the pool; on the way home*.

Hightower & Borriello (2001) also mention multiple ways with which to describe location. An *absolute location* system uses the same frame of reference to describe all located objects within in its domain. All devices that use the system must use the same frame of reference. GPS readings are good examples. All GPS devices utilise one of many coordinate standards to report location. Thus, any two objects that take GPS readings at the same physical position on Earth will have the same absolute location. *Relative* positioning systems have frames of reference that can are dynamic. Relative location is relative to other, possibly moving, objects. For example, SONAR is a system emits a sound that reflects off an object and returns to the system. The measurement of its return provides information on the location in terms of distance and direction of that object relative the SONAR device. If the absolute positions and relative locations of objects are known to an observer, then that observer can calculate their own absolute position through methods such as triangulation.

GPS has featured in this paper solely because the reader is likely to have experience with a GPS. There are various other ways in which location information can be obtained, or more appropriately, sensed.

Scene analysis, as defined by Hightower & Borriello (2001), and Cook, Buckberry, Scowcroft, Mitchell and Allen (2009), is a method by which absolute location information can be obtained. An object performing this type of analysis senses various features of the scene in which it finds itself. The object then uses those

patterns, and other existing information to infer its location. In *static* scene analysis, the existing information is a set of pattern-location mappings stored on a database. The patterns from the scene are compared to the patterns in the database to check for location matches. In differential scene analysis the object records a series of patterns, from different vantage points, within a scene that contains a feature with a known location. By tracking the changes in the object's movement around that feature, a location can be inferred.

Systems such as these are most commonly used when GPS devices do not have a clear line of sight between them and the satellites that provide them with location information. Bahl and Padmanabhan (2000) implemented a system they called RADAR that is designed to track the locations of employees inside a building (where GPS does not work). They chose to use static scene analysis, wherein the signal strength from multiple Wi-Fi access points served as the scene-specific feature-patterns. A floor plan of their working space (one floor of a building) was assigned coordinates. The database of coordinate-to-signal strength mappings was created by having a mobile host machine that pinged the APs at each of the coordinate points on the floor plan. With this system in place, users were able to use the database to look up the location of any active Wi-Fi device in the building with high rates of success. Cook et al. (2009) used a similar implementation, utilising the infrastructure Wi-Fi access points in a building, but with a lower resolution of pre-stored positions in their database. In order for scene analysis to work successfully, it does require that a database be created, which geometrically configures the working space that the system occupies. Blue-Fi (Ananthanarayanan & Stoica, 2009) also utilises scene analysis to predict being relatively local to a Wi-Fi network, as stored in a database.

Proximity sensing is a method by which relative location can be attained. Using some physical and measurable property of an object at a location, one can determine whether they are near to that object, and thus near to that location. Wireless networks that cover large areas, such as cellular networks and Wi-Fi networks with access points that are already installed throughout a building or campus, lend themselves to performing proximity sensing in this way. A mobile device can sense the surrounding access points, thereby attaining a relative (or symbolic) location with which it can describe itself.

NearMe (Krumm & Hinckley, 2004) makes use of proximity sensing and, to a degree, scene analysis. A client-server system wherein clients are devices equipped with an 802.11 capable device. The server is a database of clients mapped to relative locations over which NearMe's algorithms run. Relative locations are defined by the identities and the received signal strengths (RSS) of sensed Wi-Fi access points. The clients, who are in a physical position, sense their current relative location, and register it along with their identities on the server. In this way, a temporary scene is created at the physical location of the client. The client assumes the role of the known location, and the relative location becomes the pattern required for static scene analysis. Other clients can then perform scene analysis to check to see if they are near to this physical position. When a client leaves that location, the scene is deregistered from the server. In addition to this, NearMe allows users to register locations such as "Printer", and "Lavatory" with the server that are more permanent. The clients need only the identities and signal strengths of the access points. They are not associated with any specific network, and thus can utilise any AP that is broadcasting at the time of scanning.

These location systems provide a means by which relevant location information can be attained. Particularly, information that describes the space enveloping that location, what perhaps occupies it, and how people, the users of that space, experience it.

In summary, RADAR (Bahl and Padmanabhan, 2000), concentrates on locations within a *predefined* space, and thus is less appropriate for the dynamic nature of device discovery. However, with a system such as Blue-Fi (Ananthanarayanan & Stoica, 2009) and NearMe (Krumm, J. & Hinckley, K., 2004) information can be added dynamically to the server. This increases the utility of the system by allowing it to function in new areas without prior configuration. This dynamic is potentially very useful to the generalization of device discovery systems.

2.4 Space and proxemics

The concept of space is one that is abstract. Humans live in and engage with a variety of spaces on a daily basis, without being consciously aware of what it means to inhabit that space. Space has been studied for many years, and spatial theory is a complex field that permeates disciplines including physics, philosophy and

psychology, amongst others. For the purposes of this paper, it is important to have an understanding of what is meant when the terms *space* and *proxemics* are used.

The word space is used in many different contexts and often has different meanings for each context (Sundholm, 2007). Personal space can describe areas of proximity surrounding an individual (Hall, 1966, as cited in Hall et al., (1968); Mentis, O'Hara, Sellen & Trivedi, (2012); Sundholm, (2007)) as seen in figure 2. It can also refer to a territory that an individual owns, or has control over; something physical such as a bedroom, or something more abstract such as a private diary or online journal (Hodkinson & Lincoln, 2008). Social space refers to a physical, virtual, or abstract area. A place where individuals can gather, and can interact if they choose to do so; for example, at a dinner party, or in an elevator. One can think of it as the areas where the personal space of more than one individual intersects (Sundholm, 2007). As humans, we move through social spaces in almost everything we do (Sundstrom, 2003). Interactions in social spaces need not be concurrent. An active online forum can be defined as a social space, even if only one user is contributing at any given time. Public spaces are a subgroup of social spaces that are generally accessible by all members of the public.

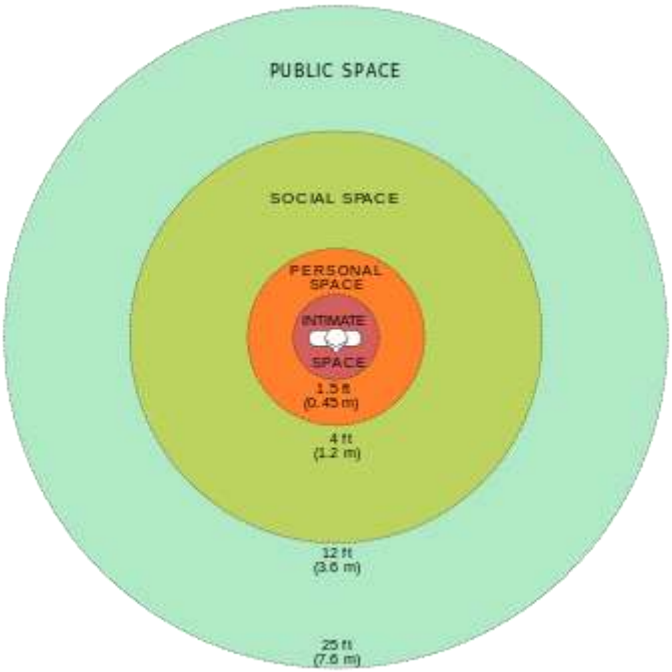

Figure 2.2. Hall's zones of interpersonal space

Crossen & Budzik (2006) discuss the ways in which public spaces are used. Some are used to promote certain types are social interaction (dinner tables in a restaurant), others have a more mechanical; made to be used by the public, but not used for social engagement (elevators). All these spaces are created, or exist, for a purpose. In general, the physical or abstract confines of these spaces limit their use to that purpose.

Distance, a measurement of proximity, is traditionally thought of in continuous values, but they can also be measured in discrete values (Greenburg, Marquardt, Ballendat, Diaz-Marino & Wang 2011). *Proxemics* is a notion put forth by Edward Hall in 1966 (Mentis et al., 2012), and refers to the study of human's perception, and the use of spaces such as those described above (Hall et al., 1968). In describing proxemics, Hall defined four zones that surround an individual human being (figure 2), *intimate zone, personal zone, social zone and public zone*. These zones measure the distance between an individual and another person (actor) by seeing whether or not the actor occupies space in one of these zones. Actors that are more intimately connected to the individual tend to fall inside more intimate zones, therefore it can be used as measure how emotionally "close" the individual and actor are. The words used to describe these measurements of space and distances are semantically related

to interpersonal relationships. This particular aspect of proxemics is relevant to computer science fields such as Human Computer Interaction (HCI) or Computer Supported Collaborative Work (CSCW) (Greenburg, 2011; Mentis et al., 2012).

The interpersonal names given to the zones are not appropriate for the purposes of this project, and neither are the specific dimensions usually associated with those zones. But the idea is important, and can be moulded to suit device discovery. Greenburg et al. (2011) apply proxemics to the field of ubiquitous computing. They speak of entities; people, digital, non-digital things and any mix of these things. They also speak of *inter-entity* distances and locations, and how to measure them. Distance here, much like Hall's zones, can be measured discreetly. Phrases describing abstract locations such as *in the same room* and *same neighbourhood* can be used instead. The system NearMe (Krumm & Hinckley, 2004) utilizes two such zones of proximity; *short range proximity* and *long range proximity*. Devices and locations within short range proximity are those that share at least one sensed access point with the client performing detection. Those within long range proximity have any number of access points between them. The client "senses" these by hopping through access points which have overlapping broadcast areas. Thus the distance between the object doing the sensing, and the object which it senses is discretely measured.

Hall's (1966) zones are of great interest, as they tie in closely with the sensing distances associated with Cellular, Wi-Fi and Bluetooth radios. In the same way that proxemics uses these zones to predict the level of intimacy between two individuals, zones relating to sensing capabilities of the different devices can be used to predict if other entities can be connected to.

In scene analysis systems, when a device establishes its location on the server, it essentially creates a space around itself. It is within this space that inter-entity distances between it and other entities can be measured. Not all entities in this space will be able to form a non-infrastructure Wi-Fi connection, thus the term *interaction space* is not a good term for this space. Rather, this space shall be known as the *Discovery space*. Much like Hall's zones of interpersonal space, the discovery space will be divided into four zones, each relating to a symbolic location, and providing a qualitative indication of inter-entity distance. Moving from outermost zone to

innermost zone, these shall be named: *same neighbourhood; same block; same building; same room.* Each of these refers to a defined area (that is roughly correlated with the broadcasting range of the devices used as features in scene analysis.) Each class of area's dimensions can differ drastically within that class. The size of a room varies depending on a large variety of factors unrelated to the classification of being a room. The boundaries of a neighbourhood are arbitrarily defined by rivers, roads and post offices. While these boundaries exist in our physical space, they are nearly non-existent in discovery space. When two devices are found at the same GPS location, but on separate floors, they are very far apart in terms of interpersonal distance However, they may be as near as three metres away from each others. They are effectively in the same room, when considering inter-entity distance. The choice of each zone name merely relates an area surrounding the client that created the discovery space that is "roughly that size (room, building, block, and neighbourhood)". The most important thing to realize is that a room is physical space within a building, which is a physical space within a city block, which is a physical space within a neighbourhood. In addition, an entity inhabiting an inner zone simultaneously inhabits the zones outside of itself

2.5 Conclusion

We have looked at the two sensors most commonly associated with extra cellular network connection on mobile device, and pointed out their relative strengths and weaknesses.

Wi-Fi is the choice network type to connect to in terms of bandwidth and latency. However the amount of power it requires can limit the utility of the device upon which a user is connecting. Bluetooth is a low-power alternative to Wi-Fi, but its current widespread version (2.1) has a significantly higher connection latency and a slower transfer rate.

We have also looked at ways in which the desirable characteristics of the radios involved in sensing can be used in a complimentary manner, essentially extending the life of the device's battery without sacrificing transfer rates.

The lesson learned, is that one should spend as little time as is possible in a high powered state. But this is not always possible if devices rely on the power consuming radios themselves in order to make contact with nearby devices. However, devices do

not necessarily need to sense each other directly in order to be made aware of each other's presence. All they need is the knowledge that other devices are in range for connection. Gaining this knowledge requires that a device be aware of its current location relative to something that it can sense; with device discovery, this is usually network infrastructure. It also needs knowledge of the location of other devices, and the distances between them and itself.

A number of systems where looked at in this chapter that make use of similar concepts RADAR, NearMe and Cook et al. (2009) used only Wi-Fi access points to aid their systems. While they may not have benefitted from using other types of access points, there is certainly room for extending similar systems with cell tower and Bluetooth access points, either in a hierarchy of power sense, or in a hierarchy of proximity sense. This is what the device discovery system in GROUT will attempt to do.

3. Design

The device discovery system in GROUT was designed with many of the ideas addressed in the background chapter. The concepts explored are potentially useful, and the GROUT system will attempt to take the best of those, and incorporate them into an efficient and functional system. This section outlines the specific requirements of the system and how the previous work enables the meeting of these requirements. It describes the structure of the system at various levels of detail and explains the roles of each component in the system.

3.1 Constraints

3.1.1 Infrastructure

The GROUT networking system is not being built with the intention of running on existing network infrastructure. However, the device discovery system will rely heavily on infrastructure to perform its function. It will function at its best in areas that have a number of cellular towers, Wi-Fi access points and Bluetooth devices. The only situation where infrastructure will not be required is if the client has its Bluetooth radio set to *discoverable*. On the Android operating system, *discoverable* must be switched on by the user, and only remains active for a limited amount of time.

3.2 Requirements

The main requirement of this section of the project is to provide the networking section with a list of devices to which it can connect; this is likely to be a list of Wi-Fi MAC addresses. It has already been discussed that common radios have discovery mechanisms built in. However, accomplishing discovery with the research question in mind places additional requirements upon the system. The research question requires that the system attempts to be efficient in terms of power consumption and monetary expense associated with data transfer.

3.2.1 Power consumption

We have established in chapter 2 that one of the best ways to reduce power consumption is to reduce the amount of time that the device spends using the high powered radios. It has already been shown that the radio which consumes the most power would be the Wi-Fi radio while in scanning mode (Abdesslem et al. 2009).

Thus, the requirement becomes to use this Wi-Fi scanning as little as possible. We have identified two methods to minimise this.

1. Create a system that reliably predicts when to power up the Wi-Fi radio, in order for it to scan for devices. This depends on hardware such as motes (Persing et al., 2005) or Bluetooth. Low power motes are not installed on common Android devices, but Bluetooth is. With this method, an option would need to be built that allows users to force the device to perform discovery, should prediction fail. This option undermines the purpose of the prediction.
2. The Wi-Fi radio need not be used to scan specifically for devices. If the device can be made aware of its location, it need only communicate with a server in order to discover devices that share the location. The Wi-Fi radio need only run so as to gain the information regarding its location.

With method one, the Wi-Fi radio must scan for every new instance of discovery regardless of whether it has changed physical position or not. The power saving comes from reducing the occurrence scans to only those occasions when a discovery is likely. With method two, the device need only scan if it wishes to perform discovery in a new absolute location. If it has not moved, then discovery can be performed any number of times without introducing the Wi-Fi radio at all. GROUT utilises method two as means to save power when performing device discovery. Using the cellular, Wi-Fi and Bluetooth radios, a device will scan for any object broadcasting on those bands. It will then send all the objects that it has detected to the server. This will be established as that device's relative location. The server will then check if that device shares a relative location with any other devices. In this way, the client need only communicate with the server to discover other devices, and does not have to perform the expensive scan again.

3.2.2 Accuracy

Saving power in the way described in the previous subsection means the GROUT system will discover device MAC addresses using an intermediary service; a service not provided by the device itself. As such there is a chance that the results returned will not be within the current range of Wi-Fi connectivity. For example, let us consider an access point in an open field. This AP can have a broadcasting range of

100m (Ananthanarayanan & Stoica, 2009). Theoretically, this means there is a circle with centred at the AP with a radius of 100m. Now consider two devices within this circle. A system such as NearMe (Krumm, J. & Hinckley, K., 2004) would find the AP to be common to both devices, and consider them to be relatively co-located. However, the phones could potentially be right next to each other, or be 200m apart, and are thus unable to form a direct connection (Figure 3.1); at least, one that does not make use of the access point as a node in the

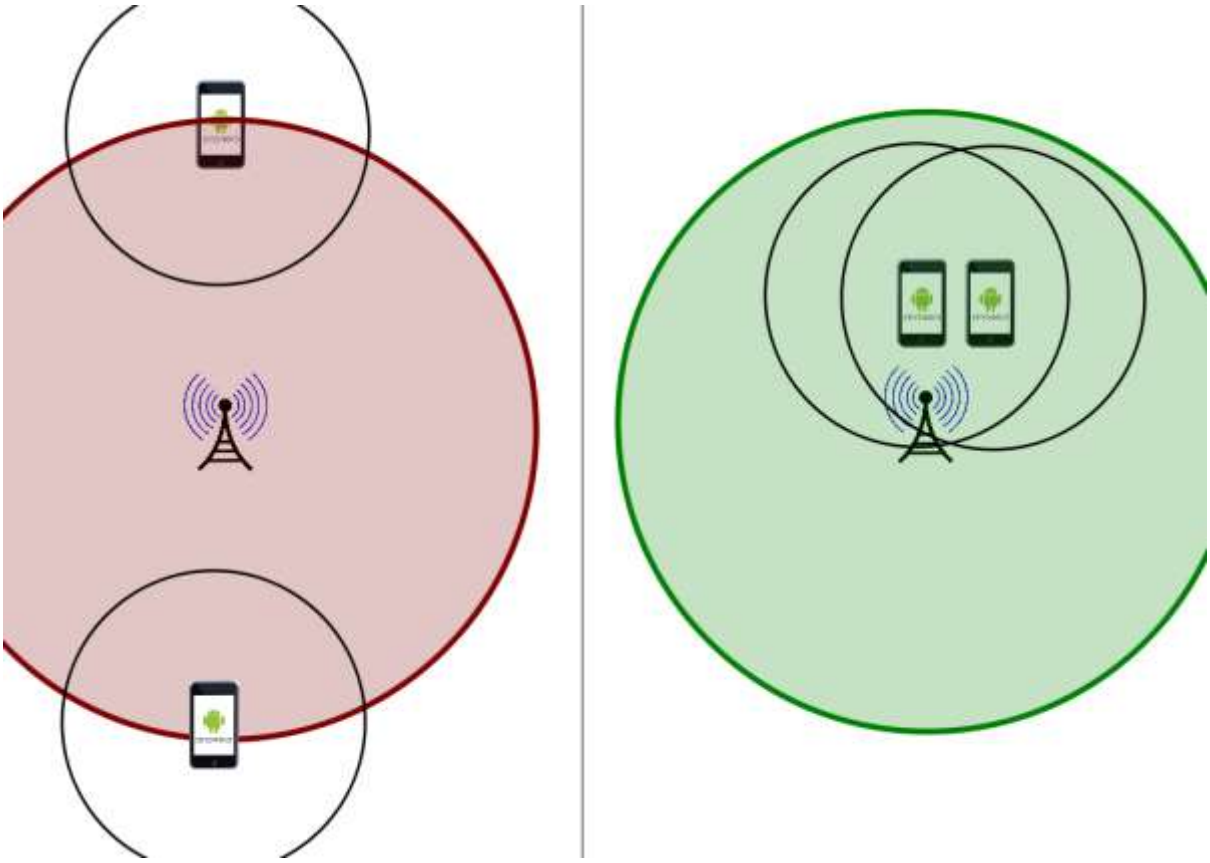

Figure 3.1. Two scenarios returning the same relative location. The left example cannot result in an ad-hoc connection and returns a false-positive. The right example can.

network. When two devices are said to be in the same location by the system, it may be that this result is a false-positive if

1. Both devices are in range of an access point, but are not in range of each other.
2. A device's relative location has changed, but this change has not been reflected in the database.

Alternatively the system may not register that devices share an absolute location when in fact they do. False-negative results may be due to

3. Devices are not in range of any access points
4. Devices are not in range of enough common access points to guarantee a connection

NearMe considered a variety of ways in which to form better estimations of co-location. The simplest of which was to sum the intersection of access points each device had in common. RADAR (Bahl and Padmanabhan, 2000), and NearMe made use of access point - signal strength relations to give each access point a weighting, and further increase the likelihood of a correct estimate. GROUT will use a the former method of increasing accuracy; summing the intersection of access points and other sensed objects

3.2.3 Low cost

Any communication with a remote server requires data transfer across a network. While, GROUT device discovery will save power by decreasing the amount of time that it spends scanning, it will increase the frequency of client server communication. As devices using this system rely primarily on Wi-Fi infrastructure to establish its relative location, it may be that devices will often be connected to a Wi-Fi network over which data can be sent. Free Wi-Fi access is increasingly common. The number of free Wi-Fi hotspots in the USA began to outnumber paid hotspots for the first time in 2010 (Tofel, 2010). The town of Stellenbosch, South Africa, is extending a free Wi-Fi network to incorporate their greater limits, and not just the town centre (Muller, 2012).

This will not be the case for every user nor every situation, and GSM or 3G data transfers are likely to occur. As already mentioned, cellular network data transfer can be expensive. Regardless of the affordability of this data, minimizing monetary cost is always of concern to users. Thus keeping the amount of data transferred to a minimum is critical to the adoption of a system such as GROUT. Even if the system manages to save power, it will be for nothing the cost of server communication outweighs the benefits gained in power conservation.

3.3 System Design

The GROUT device discovery system is made up of two compnents; a client application and a web application. The client is an Android mobile device and the web application runs on a server configured to perform the information storage, and discovery calculations. In order to perform discovery, the following procedure is followed.

1. Client collects information regarding itself.
2. Client scans for nearby access points.
3. Client registers itself and all detected access points on the server.
4. The server checks if other devices are nearby to the client.
5. The server estimates the relative distance between the client and all nearby devices and returns this information to the client.

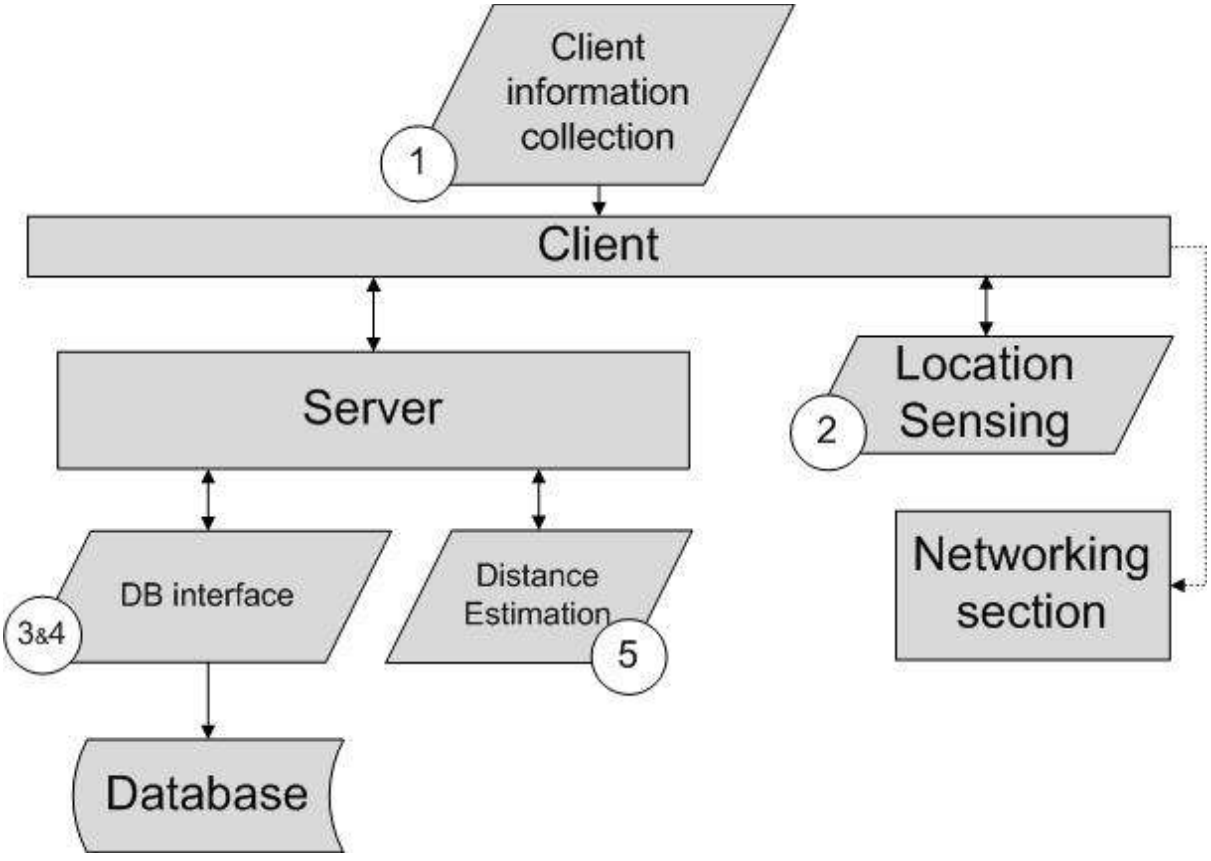

Figure 3.2. System structure overview.

Below, the design of both the client and server are discussed in greater detail.

3.3.1 The client

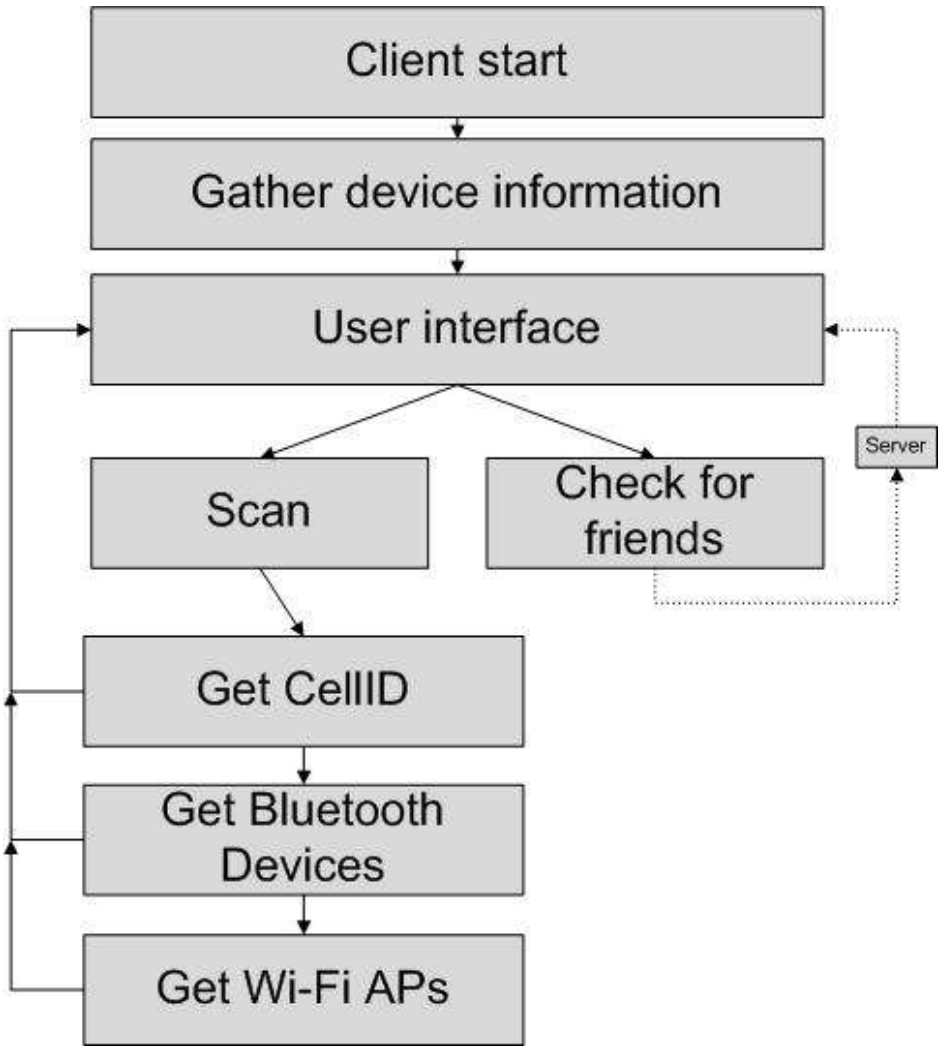

Figure 3.3. Flow diagram of client functioning.

Starting up

The client has two objectives prior to communicating with the server. Firstly, it must collect information concerning itself. It must test to see that its Bluetooth and Wi-Fi radios are activated, and activate them if they are switched off. It must then begin collecting information to send to the server.

Identifiers

There are number of possible unique identifiers that can successfully distinguish between devices. Bluetooth connection protocol requires user interaction; users are required to explicitly choose the device to which they connect. In order to assist users in identifying the device to which they intend to connect, they are able to give their

phone a name. This, however, is not guaranteed to be unique. Most devices have a unique identifier number, such as Android's secure ANDROID_ID number (Android API Reference, 2012). The device also needs to collect its own Wi-Fi and Bluetooth MAC addresses in order for the server to function correctly. MAC addresses are unique and can therefore be used instead of the other identity options.

The second objective is to collect information concerning its current location. Location in this sense is defined by the number of access points that are in range of the device. GROUT is concerned with three types of access points; cell towers, Wi-Fi access points, and Bluetooth devices. At least one of these is required by the server if device discovery is to occur.

User interface

The separation of the scanning, and client-server communication functions has already been discussed in under the heading *Asynchronousity*. In order for users to gain the maximum utility from the system, they must be provided with the agency to control when each of these tasks is performed. The user interface should be made up of two buttons, or equivalent objects, that the users may interact with in order to begin the relevant tasks.

Cell towers

In the case of cellular devices, the information for a current cell tower connection is already known by the phone. A scan will be performed to identify any other towers within range. The Cell ID will be used to as unique identifiers for each tower on the server. Identities should be stored in a list.

Wi-Fi

Wi-Fi scanning will be performed to identify the access points that are within range of the device. The actual network with which each access point is associated is of no interest. Only the MAC addresses of each access point need be recorded. These pieces of information will stored so as to be sent to the server at a later time.

Bluetooth

Bluetooth scanning will be performed in order to identify any Bluetooth devices in the area. These devices need not be other devices to which an ad-hoc network can be formed, but rather any device broadcasting a Bluetooth signal, and MAC address.

This can include Bluetooth headsets, printers and computer mice. The Bluetooth scan should be performed as early as possible, as it is the process that consumes the most amount of time (Hal, Vawdrey & Knutson, 2002; Kinney, 2003).

Asynchronousity

The entire information gathering process is accomplished independently from server communication. Performing the scans and data-transfer at the same time means that both activities must be performed for every each action. This increase power consumption and data usage unnecessarily and is not in line with the requirements of section 3.2.

Once the location information collection completes, the information is processed into a form appropriate for transfer, and sent to the server.

Server communication

As already mentioned, location scanning and server communication are asynchronous. Each can be done at any point in the use of the system. The button that begins server communication can be pressed in three situations.

1. After a scan has been performed. The client will send all the information regarding location it has acquired to the server.
2. During a scanning operation. The client will wait until the Bluetooth scan has found one device, or is complete. At either of these points, it will send information to the server.
3. Before any scanning has taken place. The client will register itself on the server, but will not have any location information besides its own Bluetooth MAC address. This is essential for the situation in which there is no infrastructure to estimate location. If another device can sense the client's Bluetooth signal, then the GROUT system will allow them to discover each other.

The server returns a list of devices that share a space with the client. The list contains only the Wi-Fi MAC addresses, and the estimate of discrete distance between the client and that MAC address, which is presented as a symbolic location.

3.3.2 The server

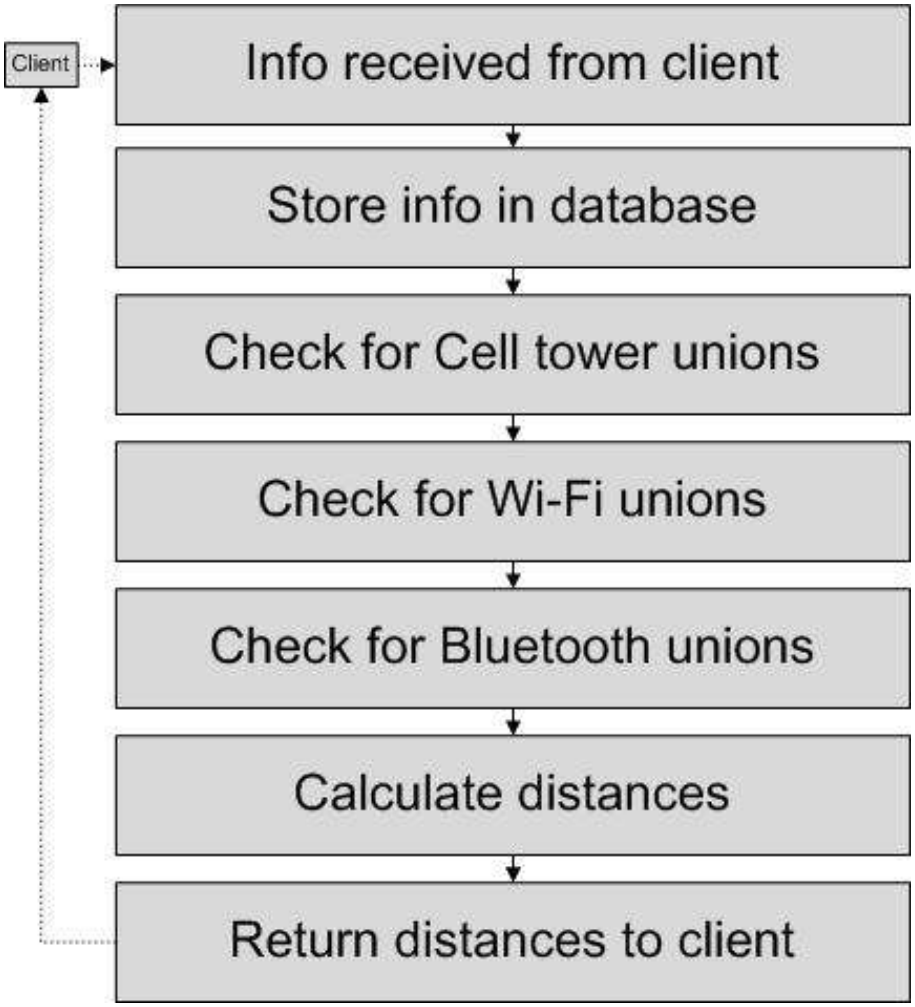

Figure 3.4. Flow diagram showing server work flow

The server is where the scene analysis takes place and inter-entity distances are calculated. The server must communicate with a number of clients, receive their relative location by way of access point information and store it in a database. It must be able to manage the information, and use it to perform the necessary calculations that return an estimate of the inter-entity distances.

Executable code

The server provides a service to the clients. As such it must be able to execute scripts, or otherwise some programme that can accomplish tasks that are described below. It needs to be able to parse a file that it received from the client and store it data-structures. It must be able to perform various calculations using those data-structures. It must be able to interface with a database to both store and retrieve

data. It must be able to generate a file of the results of its calculations into a format that is both readable by the Android operating system, and is not large in terms of data size.

Database

A database is a crucial component in performing scene analysis (section 2.3). It stores the contexts against which all other clients compare their own context. Therefore, the database must keep track of one instance of each device and its information registered on the server, along with every cell tower, Wi-Fi access point, and Bluetooth device that is in that device's discovery space. In a relational database, this requires that the database has the following tables:

- Devices: The data stored in this must include a unique identifier for the device, its Wi-Fi MAC address, its Bluetooth MAC address This relation must store only one instance of a device at any given time.
- Cell Towers: An entry for each cellular tower sensed by each device must be stored in this relation. Tuples should consist of the device's unique identifier and the CellID of the tower.
- Wi-Fi APs: An entry for each Wi-Fi AP sensed by each device must be stored in this relation. Tuples should consist of the device's unique identifier and the MAC address of that access point.
- Bluetooth devices: An entry for each Bluetooth device sensed by each device must be stored in this relation. Tuples should consist of the device's unique identifier and the MAC address of that Bluetooth device.

Only the one location for each device may be registered at any given time. This ensures that each device represents only one discovery space, and no ambiguity can occur. Therefore, whenever a new instance of a device is registered, every tuple in every relation containing that device's unique identifier must be deleted and the new context inserted to the relevant relations.

Calculations

GROUT calculates the unions of access points between different devices. It uses this calculation to estimate the inter-entity distances. Inter-entity distances are measured discretely as follows: When a devices registers on the server, it creates a discovery space.

The three types of access points that a GROUT client senses indicate different levels of relative location; the inter-entity distance.

Cellular towers provide a very coarse estimate of location. With a broadcasting range of 500 m, the client could be anywhere inside a circular area with a diameter of 1km. A shared cell tower between two devices indicates a high likelihood that the two are in an area that could be the size of a small neighbourhood. This measurement also places an entity in the outermost zone of the client's discovery space; the *same neighbourhood* zone.

Wi-Fi access points provide information regarding the location of an entity in the distances between outermost and innermost. They of have a broadcasting range of up to 100 metres (Pering et al., 2006). It is likely that this range will be smaller, as there are a number of factors that interfere with signal strength, including Bluetooth (HP, 2002) and the direction in which the client device is facing (Bahl and Padmanabhan, 2000). As the number of common Wi-Fi access points between two entities increases, so does the likelihood that the inter-entity distance is smaller. Wi-Fi is the source of location information that places entities in the middle two distance zones. There exists an interpretation of the telecommunications law in South Africa that states that *unlicensed* wireless networks are forbidden from being broadcast across public roads (Telecommunications act, 1996; Cull, 2009). All wireless networks must be contained within a city block. Now while this may not hold true forever, or at all in other countries, it does suggest a neat way of describing the two distances representing relative location by Wi-Fi. A city block is defined here as being an area surrounded by public roads. The outer zone is called s*ame block*, and requires that the two entities hold at least one Wi-Fi access point in common. City blocks that have Wi-Fi access points can have buildings on them. Thus, any entity occupying the inner zone of the Wi-Fi discovery space is said to be in the *same building* as the client entity. The number of common Wi-Fi access points between the two entities is described after the next paragraph.

Common Bluetooth devices provide the most fine-grained estimation of location. Bluetooth radios on most devices have an effective broadcasting range of roughly ten metres (Ananthanarayanan & Stoica, 2009). Two entities within range of the same Bluetooth device should therefore be, at most, within 20 metres of each other,

although this distance is likely to be smaller. The two entities are likely to be in the same room as each other. Thus from unions made up of at least one Bluetooth device, entities are measured as being within the innermost zone of the discovery space are found; these entities are said to be in the *same room*.

The algorithm for calculating the inter-entity distance produces "closest known" distances, which is defined as follows:
The most accurate inter-entity distance that the system can guarantee.
For example, if two devices are in the same room, and the only access point they have in common is one cellular tower, then the closest known distance that a system can give is *same neighbourhood*.

All zones, except for the *same building* zone, require only one instance of a common feature in order decide what two device's inter-entity distance is. In the absence of a Bluetooth device, estimating that two entities are in the same building as a closest known distance requires a number of Wi-Fi access points that is greater than one. Exactly how many are required before the threshold between *same block* and *same building* is crossed is relative to the physical location of each Wi-Fi access point, as can be seen in Figure 3.5 and 3.6. Literature relating to these numbers has yet to be found. For the purposes of this project, the threshold number has been set at any number greater than one.

Considering thresholds, a number of common cellular towers between two entities must exits, where the threshold between the *same neighbourhood* and *same block* zones are crossed. The same can be said for the number of common Wi-Fi access points and the threshold between *same building* and *same room*. As situation must exist where a number of Wi-Fi towers will provide the same quality of location information as one Bluetooth device. We intuitively know this, because we already have in this project a hierarchy of devices providing different inter-entity distances. What we do not know is how each device is weighted. A very naive calculation of what the weightings might be is presented in table 3.1.

Table 3.1

The area of each access point based on the given radius, and the ratio value of the probabilities of selecting a square metre from each area

	Radius	πr^2	$\frac{1}{Area} \times 10^6$
Cell tower	500	785398.16	1
Wi-Fi AP	100	31415.927	31
Bluetooth	10	1256.6371	795

As can be seen from this, the numbers are possibly quite large. Again, a thorough analysis of the physical situations of access points are required to find more accurate numbers; an undertaking outside the scope of this project. Nonetheless, what this calculation produces, is an estimate of how many lower tier access points are required to predict with the accuracy of the tier above. It would take roughly 31 cellular towers to provide the same level of location detail as one Wi-Fi access point. And it would take roughly 25 Wi-Fi access points to provide the same location detail as one Bluetooth device. These values will be used to provide threshold values at which the lower tier access points, such as cellular tower, can produce a higher tier result, such as *same block*.

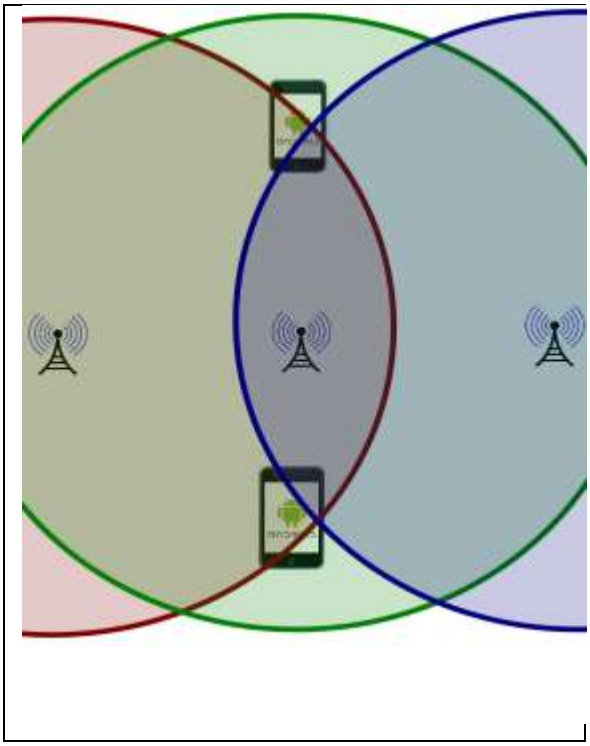

Figure 3.5. Two devices in the same building, each sensing three Wi-Fi APs

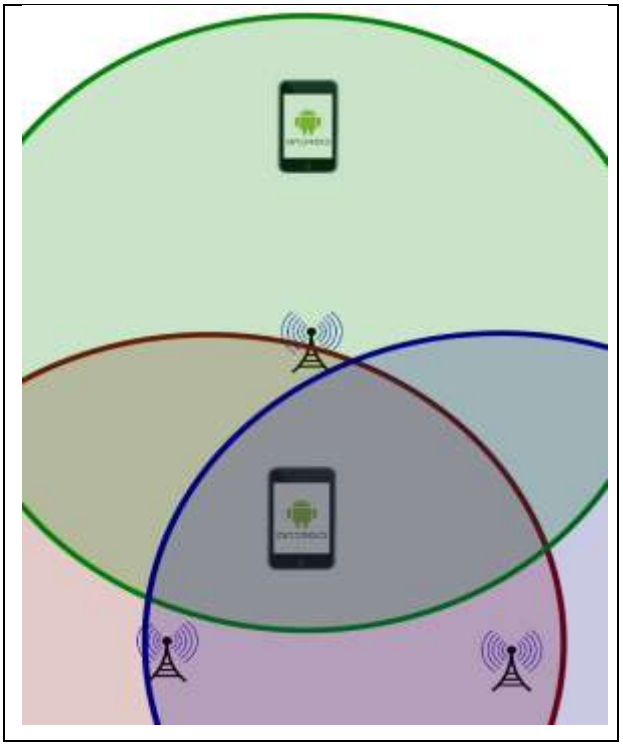

Figure 3.6. Two devices in the same building, one sensing three APs, one sensing only one AP.

The algorithm for calculating the inter-entity distance is as follows in Algorithm 1.

3.4 Summary

In this section, the requirements for the system have been detailed. Each component, namely the client and the server, have been defined in terms of purpose, and a general the information required to create each component has been outlined. In the next chapter, the design will be applied to mobile devices and a server machine in an attempt to implement the GROUT device discovery system.

Algorithm 3.1 : Calculating inter-entity distances

Client = device that called this instance of server calculation.
uniqueID = identifier unique to devices
value = inter-entity distance
cellTower = list of cell towers sensed by *client*
wifiAP = list of Wi-Fi access points sensed by *client*
btDevice = list of Bluetooth devices sensed by *client*
listOfDevices = new array[*uniqueID]*

For all *cellTowerID* in *cellTower* **do**
    devices = Select from database all devices that share this *cellTowerID*
    For all *device* in *devices* **do**
        listOfDevices[device] = "same neighbourhood"
    End for
End for
For all *AP* in *wifiAP* **do**
    devices = Select from database all devices that share this *AP*
    for all *device* in *devices* **do**
        If (*listOfDevices[device]* = "same neighbourhood")
            listOfDevices[device] = "same block"
        else if (*listOfDevices[device]* = "same block")
            listOfDevices[device] = "same building"
        End if
    End for
End for
Devices =Select from database all devices that sensed client device's Bluetooth
For all *device* in *devices* **do**
    listOfDevices[device] = "same room"
End for
For all *bt* in *btDevices* **do**
    Devices = Select from database all devices that share this *bt*
    For all *device* in *devices* **do**
        listOfDevices[device] = "same room"
    End for
End for

4 Implementation

In the previous section, the design of the GROUT discovery system was discussed in some detail. This chapter deals chronicles the process of performing an implementation of the design presented in the previous chapter. It deals with the android platform and the APIs that are most important to this system. It looks at the set up of the web service. The design was adhered to as much as possible, however certain limitations dictated that the implementation diverge, and be adapted to suit the available devices and platforms. These limitations included the implementation platform, and a compatibility issues with cellular tower API

4.1 Client

4.1.1 Windows Phone 7

It was the researchers' intention that the entire system be built upon WP7 devices. Applications for WP7 are developed using two Microsoft frameworks; Silverlight and XNA. Silverlight is an event driven framework that makes use of XAML, a variant of xml aimed at application development. XNA is primarily used to develop loop-based games and other interactive applications. Each relies on C# to perform back-end computations (Microsoft Windows Phone 7 Development Centre). The WP7 developer SDK is designed to run within these frameworks only, and is the only official SDK available to independent developers. There are a number of problems associated with the SDK, the biggest of which is that the API is severely limited in terms of what it lets developers do. For instance, developers do not have access to a whole range of hardware functionality including connectivity radios and file system access (Rubino, 2011; Lacey, 2012; Calum, 2011), which are two functions important to GROUT. At the time that the project was being formulated, Microsoft had hinted that they were thinking of releasing a more complete SDK based on developer suggestions (Rubino, 2011). However, the addition of more complete Bluetooth API had been under review since December 29, 2011 (Reed, 2011) with no convincing indication that it would be released before the official release of Windows 8.

Despite these restrictions, some third party developers have managed to hack their way into the hidden APIs on a limited number of Windows Phones. Examples include DFT and their Bluetooth File Transfer application (Rubino 2012). Heathcliff74 (2012) developed an SDK that can run native C++ code, and access the

root file system. While these may have helped the GROUT team accomplish our goals on WP7 devices, doing so would have been outside the scope of the project. For this reason, it was decided that the interface development would remain on WP7, while the computational sections of the project would be implemented on the Android mobile operating system instead. The APIs and hardware access that are required are known to exist on this platform (Android Developer Guides, 2012a). While using Android to build the GROUT system means that the tiling analogy begins to break down. However, it does mean that the discovery and networking subsystems are more generalised to the current Smartphone market; Android accounted for nearly 70% of the Smartphone market during the second quarter of 2012 (IDC, 2012).

4.1.2 Tools

All Android development was performed using the Android SDK and Eclipse, an open source IDE that is commonly used for Android development. The Android Development Tools is a plugin for Eclipse creates an environment in which development is supported by features such as in-edit compiling that checks for syntax errors as well as GUI for debugging. The Android SDK utilises the Java programming language. As such, the Android SDK requires the Java JDK. It should be noted that developing for Android is not the same as developing for Java because Android employs only a subset of the Java API.

The devices on which the system was implemented are as follows in Table 4.1

Table 4.1

Devices used for implementation and evaluation

Name	Android Version	Maximum API Level
Samsung Galaxy Ace	Gingerbread (2.3.6)	10
Samsung Galaxy Gio	Froyo 2.2	8
Huawei Ideos	Froyo 2.2	8

4.1.3 Android Operating Systems and levels

There are a number of different versions of the Android operating system. Each is designed to run on mobile devices of differing hardware capabilities. Currently,

version 2.3 is the most common. Codenamed *Gingerbread*, the version is used by a cumulative total of 55.8% of android devices (Android Developer Guides, 2012b).

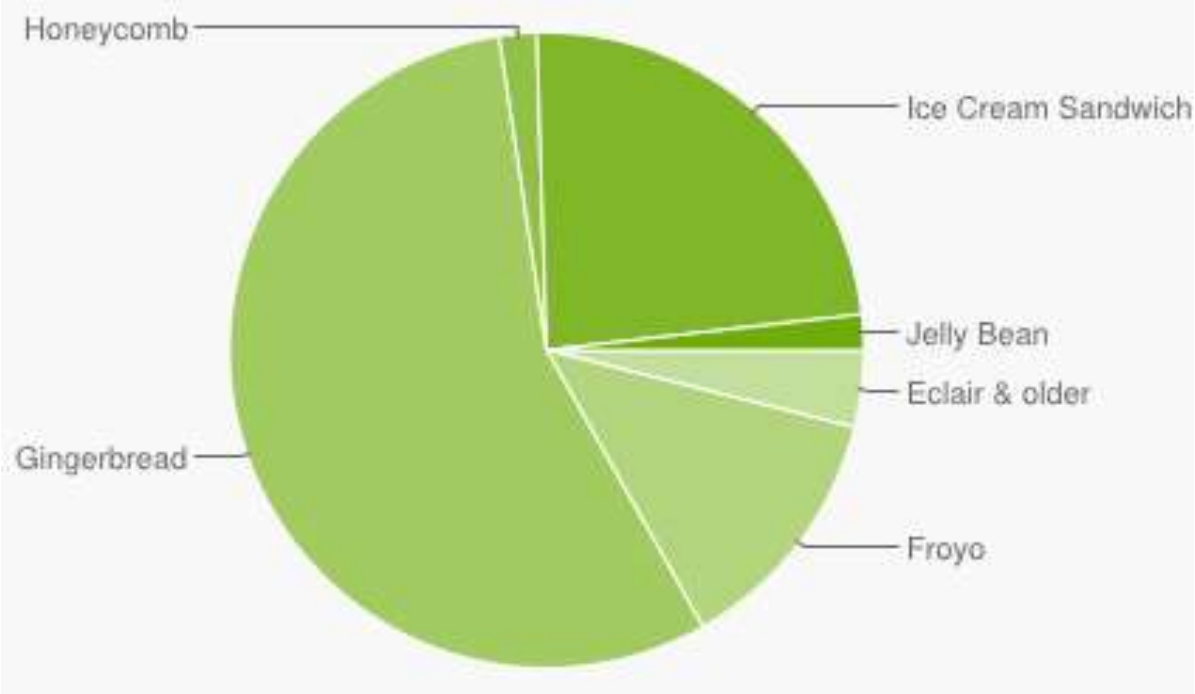

Figure 4.1. Distribution of Android versions

When developing for Android, one must keep in mind the version for which the intend to develop. This is largely due to the fact that not all of the SDK's API will function on all the versions. The API is divided into levels, which map the OS version numbers; not codenames. For example, version 2.2 (*Froyo*) uses API level 8. *Gingerbread* versions 2.3- 2.32 use API level 9 and versions 2.3.3-2.3.7 use level 10 (Figure 4.2). In general, applications are forward compatible, but not backward compatible. This is to say that an application developed for API level 10, can run on most version that can support higher level, but not the other way round. The Android API reference provides a neat filter that greys out the classes, methods and variables. The Eclipse IDE requires developers to define a minimum API level. This allows the environment to warn developers if they use an API that is in a higher level. The client system was designed with API level 8 as its target level ensuring that it would run on all the development devices (table 4.1), and 95% of all android devices.

Version	Codename	API	Distribution
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.4%
2.1	Eclair	7	3.4%
2.2	Froyo	8	12.9%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	55.5%
3.1	Honeycomb	12	0.4%
3.2		13	1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	23.7%
4.1	Jelly Bean	16	1.8%

Figure 4.2. Table showing Version breakdown.

4.1.3 JSON vs XML

In order for data to be transferred between devices and servers, a format is needed that is readable by both parties. These formats are commonly known as data-interchange formats. Two data-interchange formats commonly used on the internet are XML and JSON. Android provides API to use either of these. An XML file is made up of any number of elements, each of which can contain attributes, text data or child elements. XML files must conform to very strict formatting rules. Most files are accompanied by an XML schema, which describes that XML file. For reasons such as these, the XML format has come to be associated with with what is called the data explosion; the inclusion of data that is not essential to the data being sent. JSON is an acronym for JavaScript Object Notation and files are made up of objects in key-value pair formatting. This formatting can be easily mapped to the structure of relational database tables, something GROUT makes use of. JSON files also tend to be smaller in terms of data size than XML files (Ableson 2010). Minimizing data transfer rates are one of the key requirements of this system, and thus JSON is the format chosen for the GROUT device discovery system.

The building blocks of JSON files can be seen in Figure 4.3.

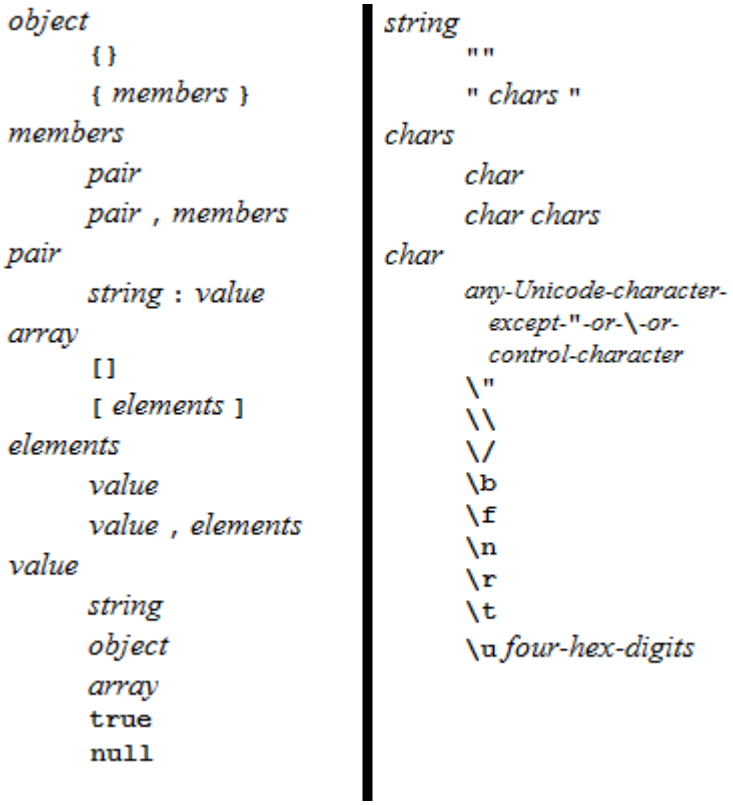

Figure 4.3. Abridged set of JSON building blocks (json.org, date unkown).

An example of a JSON file generated by the system can be seen in Appendix A.

4.1.4 Android SDK and the GROUT client system

Core of an application

Android applications have three core components. *Activities* and *broadcast receivers* are the two that are of concern here. Activities are the backbone of any Android application. One activity is one screen that has a user interface, either providing information to the user, or allowing the user to interact with the application. GROUT device discovery makes use of a single activity.

Intents are objects activate any of the three core components mentioned above. One can think of them as the object representing the application's *intent to do something else*. These intention objects can store additional information that can be used by the action invoked. For example, they can store text from one activity, and display it in the activity that the intention started.

An activity has a specific lifecycle that it must run through before it is in a state that can be said to be "running." Figure 4.4. shows a section of this life cycle that is important to implementation. The classes shown here are used in discussion later in section 4.1.5

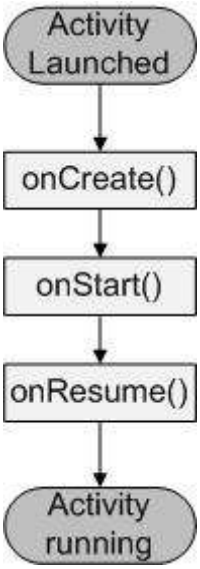

Figure 4.4. Start-up protocol for Android Activity

Broadcast receivers take action when they become aware of a system wide broadcast, such as an alarm, or a notification that a Bluetooth device has been found. This project does not make use of services, and thus they are not addressed.

Buttons

Buttons are event driven. Each button has a listener attached to it, which "listens" for a button press. When it "hears" a button press, the listener calls the onClick() method with which it associates.

Toast

Android has a neat pop-up notification API called *Toast*. This is a dialog that can display text messages to the user, and pops up over the current activity. Consisting of only one line of code, it is used frequently by the GROUT client system to convey messages to the user It has limitations, in that it cannot be called in all situations, for example, *Toast* does not work inside a *BroadcastReceiver*,

Applied to the client

The GROUT client system is made up of one activity, called the MainActivity, which contains all the client functionality. Toast dialog boxes communicate hidden functions to the user, if necessary. Two buttons are used to register the events associated with scanning, and with server communication. A BroadcastReceiver is used to listen for when the a Bluetooth device's broadcast is sensed, or when the Bluetooth scanning procedure finishes

4.1.5 The application

Startup

The activity launches and displays the screen seen in Figure 4.5. For the purposes of development, all the device and location information required is displayed on the screen. This display is not required for the successful execution of the system. *Phone Name* is the human readable identifier assigned to the Bluetooth radio. *Phone ID* is the device's ANDROID_ID as collected through the *Settings.Secure* API. *Bluetooth ID* is the Bluetooth MAC address, and *WIFI ID* is the Wi-Fi MAC address. For the sake of human readability during development, the unique identifier used across the system was the phone's Bluetooth name. This can very easily be changed to any of the *Phone ID, Bluetooth ID* or *WIFI ID*. Due to them all being represented as Strings, no significant implementation changes are required. *Data Transfer* is included for the sake of monitoring the amount of data used after each communication with the server. The *Scan* button begins the location sensing function. All results returned by this are displayed below the button. *CellID, Lac, Mcc* and *Mnc* are all associated with cellular towers. *CellID* is the unique identifier for a cellular tower, and is used by the system. *Lac, Mcc* and *Mnc* are codes associated with location, country and network operator reapsectively. Their inclusion is an artefact of a brief attempt to attain additional location information from individual cellular towers. The *Check for friends* button begins communication with the server so as to register the device, and perform scene analysis.

The onCreate() method contains code that checks to see if the Bluetooth and Wi-Fi radios are switched on. It also creates objects of the three radios that are used by this system. The *TelephonyManager* represents the Cellular radio, the *WifiManager* represents the Wi-Fi radio and the *BluetoothAdapter* represents the Bluetooth radio.

Each of these classes contains methods with which the required information can be collected.

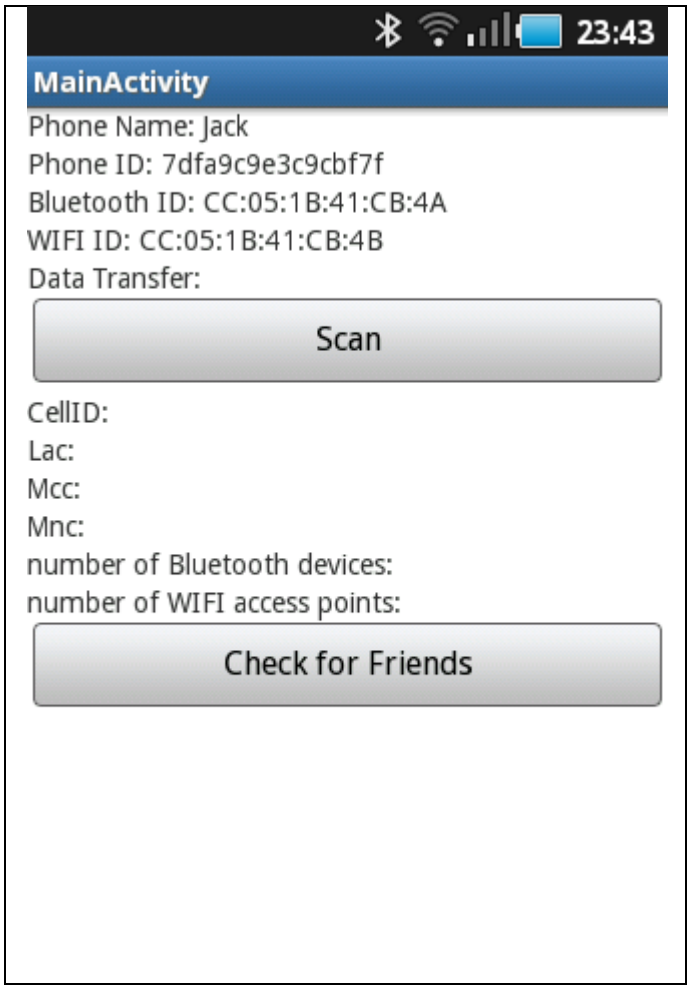

Figure 4.5. Screenshot of application when opened.

The basic code structure of the method is as shown in Algorithm 4.1.

Algorithm 4.1. *onCreate() method pseudo code.*

```
onCreate(){
  initialize radio objects
  if radios not on do
    request radios switch on
  end if
  get device unique identifier, phone name and Bluetooth/Wi-Fi MACs
  register BroadcastReceivers for Bluetooth scanning
  instantiate buttons
}
```

Bluetooth and Wi-Fi are not essential to the running of the application, but the application cannot perform optimally if those two radios are switched off. If the radios are not activated, the screen shown in Figure 4.6 is displayed upon start-up.

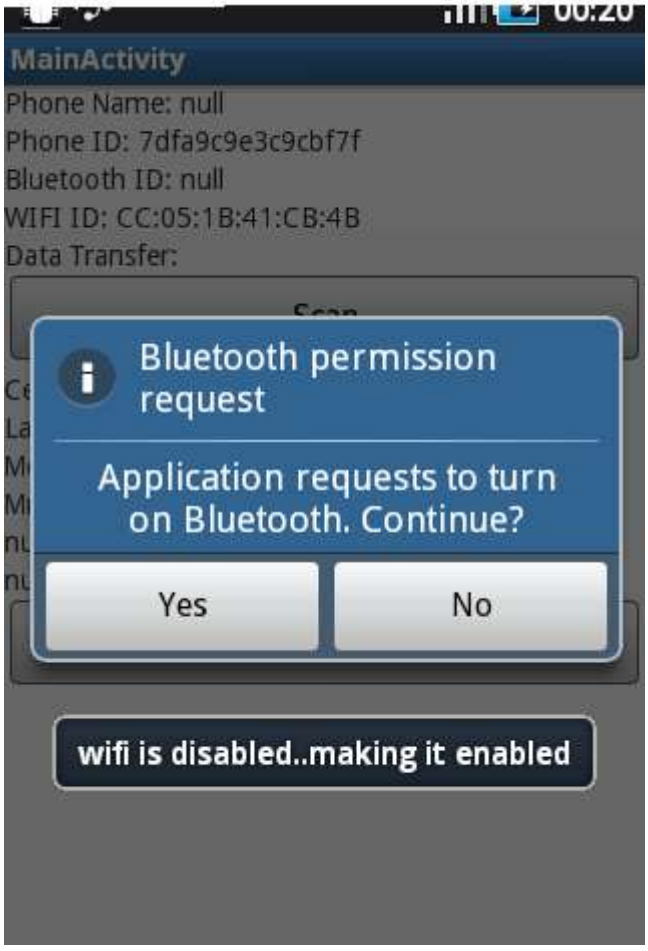

Figure 4.6. Enabling Wi-Fi, as reported by a Toast dialog, and requesting Bluetooth.

The application automatically switches on Wi-Fi, and a *Toast* message informs the user of the Wi-Fi change. The *BluetoothAdapter* asks the user if they would like to activate Bluetooth. The Android API dictates that this is the only means by which Bluetooth can be activated. The main screen is then shown is in Figure 4.6. The only difference being that the display for *Phone Name* and *Bluetooth ID* remain *null* even if Bluetooth has been switched on, and the variables have been initialized. The display is updated when the scan button is pressed.

Scanning

Pressing the scan button calls the onClick() method associated with scanning. Basic code structure for this method can be seen in Algorithm 4.2. The implementation diverges from the design here due to an apparent incompatibility between the Android *Telephony* API and Samsung devices. The methods in the *NeighboringCellInfo* class is used for obtaining the information regarding cellular towers other than one to which a device is currently connected. While attempting to access this information during implementation, all methods in this class returned negative and null values. Some research returned similar problems concerning Samsung devices (FabioCreativity, 2012; Kerts, 2010; Spike66, 2011). Some of these implied that this was a known issue (FabioCreativity, 2012; Haylicek, 2011), but no official indication of this has been found. One Prof. R. Koenig claimed that he has verified this class works on many devices, but not the Samsung Galaxy S running two different Android versions (Koenig, 2010). Given that two out of three testing devices are Samsung divices (Table 4.1), attempts to implement the local cell tower retrieval was abandoned. Although the Huawei could have potentially been able to utilise *NeighboringCellInfo,* the number of cell towers in common between it and the Samsungs would have always been, at the very most, one. Thus, only the currently connected cell tower's information is used in the device discovery algorithm.

Algorithm 4.2. *Scanning version of the onClick() method*

```
onClick(){
  Get connected cell tower information.
  If Bluetoth radio currently scanning do
    stopBluetoothScan()
  endif
  Clear lists storing Bluetooth devices and Wi-Fi access points
  Call method  startBluetoothScan()
  Begin scan for Wi-Fi access points
  Store results for each scan in respective lists
  Calculate list sizes.
}
```

The Cellular tower information is acquired through a simple method call in the *telephony* API. To acquire the Wi-Fi access point list is also simple. The *startScan()* method is called, and the results are instantly stored in memory where they can be accessed, by another method call. For Bluetooth, the procedure is slightly more complicated. The scanning process can take between ten seconds, and half a minute (section 2.1.2) and thus a check must be performed to see if a scan is in progress. Other Bluetooth devices are discovered when their broadcasts are received by a BroadcastReceiver. It is within this receiver that discovered devices are stored in the data list data-structure. Discovered devices are added to the list in real time. It is for this reason that any Bluetooth scan must be stopped and the Bluetooth list must be cleared before starting a new scan, as this ensures that if the client device's physical position changes significantly, devices for the old position do not remain in the list.

All but the *number of Bluetooth devices* fields update immediately, for time reasons described in the paragraph above Cell tower and Wi-Fi lists are filled and returned instantly, but the Bluetooth list is built, and returns, over a substantially longer period of time. The *number of Bluetooth devices* field displays *Scanning...* until a device if found, or until the scanning process completes, as can be seen in Figure 4.7.

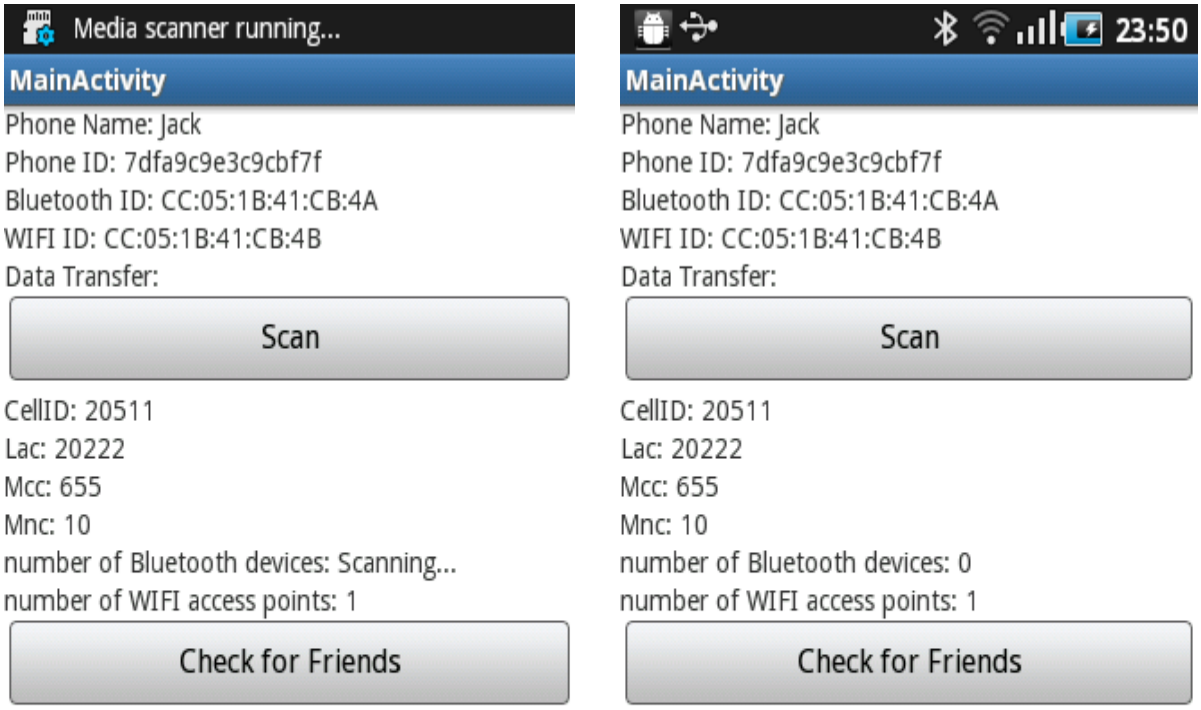

Figure 4.7. The screen after the scan button has bee pressed. Scanning is displayed until a Bluetooth device is found, or until scanning finishes.

Communicating with the server

Pressing the *Check for friends* performs two functions. It formats the location information into a JSON file, which it can then send to the server. Java provides an API that simplifies the formatting procedure. The application waits until the Bluetooth scanning finds a device, or finishes scanning before performing formatting procedure. The JSON file specification can be read in Figure 4.8.

```
JSON{
  "name" : "Phone Name",
  "uID": "Phone ID",
  "btID" : "Bluetooth ID",
  "wifiID" : "WIFI ID",
  "cell" : "CellID",
  "wifiSize" : "numberOfSensedAccessPoints",
   "wifi" : ["Wifi MAC address 1", "Wifi MAC address 2", ...., "Wifi MAC address x"],
  "btSize" : "number of sensed devices",
  "bluetooth" : ["BT MAC address 1", "BT MAC address 2", ...., "BT MAC address x"]
}
```

Figure 4.8. JSON file specification.

The second function is to create an http connection with the server using Java's Apache API. The choice of using Apache is discussed in section 4.2. Once the connection is establish, the JSON file is posted to the server. The server responds with a JSON file in turn. This contains the name and inter-entity distance of each device in the discovery space. The application reads this data, and prints it out as can be seen in Figure 4.9. Once again, it should be emphasised that the *Phone Name* is used for the sake of readability in testing, evaluation and demonstration. By merely returning the Wi-Fi MAC address instead of the name.

Figure 4.9. Inter-Entity distance displayed.

4.2 Server

The server is where all the device discovery and inter-distances are calculated. For this project, the server side functioning was implemented on the server space provided to each student.

4.2.1 Server choice

The University of Cape Town provides its Computer Science students with a LAMP style workspace on their servers. This workspace is known as *nightmare*. LAMP is an acronym for Linux, Apache, MySQL / MariaDB, Perl / PHP / Python. These are a collection of software that together provide the essential tools required for hosting a webpage or webservice. *Nightmare's* particular collection features PHP and MySQL. These two platforms will be used as the executable code and the database for the GROUT device discovery system. A private server, *Skurro,* was built and configured with LAMP, with the intention of using it for the server side of the system. However, this provided no benefit over using *nightmare*. In fact, routing and DNS complications made the remote use of *Skurro* difficult. *Skurro* was abandoned early in the implementation process in favour of *nightmare*.

PHP: Hypertext Processor is a scripting language that is very commonly used on websites and in web-services. It can read in and parse data in the form of a JSON file, as well as reformat data into a JSON file. It provides a means with which to interface with the MySQL database

MySQL is a relational database management system. It stores it data in tables. Data is added, accessed and deleted from these tables using SQL statements.

4.2.2 Database

MySQL is a relational database. As such, the data will be stored in tables that are in some way related to one another. As defined in the design, section 3.3.2, the server must register each device and keep track of Cell towers, Wi-Fi access points and Bluetooth devices for each device. Thus, the following table are required:

- Devices: The data stored in this must include a unique identifier for the device, its Wi-Fi MAC address, its Bluetooth MAC address This relation must store only one instance of a device at any given time.

- Cell Towers: An entry for each cellular tower sensed by each device must be stored in this relation. Tuples should consist of the device's unique identifier and the CellID of the tower.
- Wi-Fi APs: An entry for each Wi-Fi AP sensed by each device must be stored in this relation. Tuples should consist of the device's unique identifier and the MAC address of that access point.
- Bluetooth devices: An entry for each Bluetooth device sensed by each device must be stored in this relation. Tuples should consist of the device's unique identifier and the MAC address of that Bluetooth device.

A diagram of the tables and their properties as they were implemented can be seen can be seen in Figure 4.10.

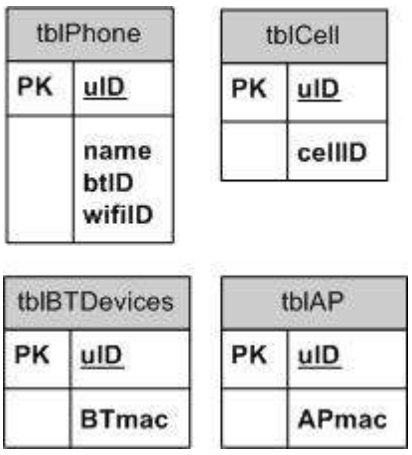

Figure 4.10. Tables in the database

4.2.3 Server side code

The server side code is responsible for parsing the JSON file received from clients, communicating with the database, and calculating inter-entity distances and returning the this measurement to the client who initially requested it in the form of a JSON file.

Registration

The registration section of the script requires that the device's identification information be parsed from the JSON file into a format that can be manipulated by the script. PHP provides a powerful API for reading JSON files. The sequences in which the objects are placed into the file are not of consequence, as each value can be

accessed directly by its key. This process involves getting the raw data that was posted by the client, and decoding it into a JSON data-structure. One can then initialize variables and store each piece of data in the JSON file like this:

$phoneName = $jsonDataObject -> name;

Where $*phoneName* is the variable name, *$jsonData* holds the json file and *name* is the key part of the key-value pair in the file.

Once the values are stored, a connection to the database must be opened. PHP provides a simple means of doing so. Once opened, any previous entry in *tblPhones* by this device must be deleted for the sake of consistency. Only once this has occurred can the new entry be inserted to tblPhones.

The code structure for this section can be seen in Algorithm 4.3.

Algorithm 4.3. *code for storing client information*
Get data from POST Decode data to JSON structure Get client info from JSON Open MySQL connection Delete previous entries for this device Insert device and information to tblPhones.

Establish location

Now that the system knows which client it is dealing with, it can begin to store location data. To do this, the PHP has three methods; one for each radio's information. Each method has the same basic structure, as can be seen in Algorithm 4.4. It loops through the JSON array associated with its particular radio and stores each value in an array. This array is used later in the script. The realevant table is cleared of any entries associated with the client, before looping through the array again inserting each sensed item into the table. Due to the possible *NeighboringCellInfo* limitation on Samsung devices, as mentioned in section 4.1.5: *scanning*, the method for cellular towers is reduced to a simple delete and insert statement over *tblCell*.

Algorithm 4.4. *Psuedocode for storing the client's relative location*

```
For all elements in JSON array do
  Array element = JSON element
End for
Delete from table where uID = client ID
For all array elements do
  Insert into table uID and element
End for
```

Local Entities

The next job for the script is to locate all the devices that fall into the client's discovery space. The algorithm for doing so was presented in Algorithm 3.1. The algorithm that was implemented differs slightly from this, due in part to the *NeighborCellInfo* tower limitation. The other deviation does not affect the results, but is, none the less, a deviation. Each radio is given a weighting: Cellular radios are weighted 1, Wi-Fi access points are rated 2 and Bluetooth devices are weighted 50. These weightings were calculated using the ratios calculated in table 3.1 in Section 3.3.2. Each entity within the client's discovery space receives a score that is made up of the number of each location feature that they have in common with the client, multiplied by the weighting associated with the feature. These numbers are based on the rough estimation performed in table 3.1. Due to the *NeighborCellInfo* tower limitation, there can never be more than one cellular tower in common, thus Wi-Fi weight is safely set at 2. Table 3.1 shows that roughly 25 Wi-Fi access points are needed to provide the same amount of location information as one Bluetooth device. Wi-Fi access points are weighted with a value of '2' and thus a value of 50 is assigned to Bluetooth devices. The Algorithm calculates total scores for each entity, and then conditional statements dictate which inter-entity distance they fall into, as seen in table 4.2.

Table 4.2
Threshold values for each inter-entity distance.

Distance	*Same neighbourhood*	*Same block*	*Same building*	*Same room*
Score (x)	x=1	x=2	2< x <49	x>49

This is a simple means of implementing the algorithm, while including a chance that Wi-Fi access points alone can result in a *same room* distance estimation. The probability for a *same room* estimation when inter-entity distance is physically only *same* is at most the same as the probability for underestimating inter-entity distance.

Returning inter-entity distances to the client

The final task that the server must perform is to format the inter-entity distances as a JSON file. The specification for the return file is smaller than the received file, and can be seen in Figure 4.11. Note that the *friends* array contains JSON objects, not merely JSON values. Again, for the sake of readability, the name is returned to the client. To change this to the MAC address, one need only use the variable storing that device's MAC address.

```
JSON{
  "friends" : [{"name" :  "name 1", "distance" : "inter-entity distance 1"} ,
               {"name" :  "name 2", "distance" : "inter-entity distance 2"}]
}
```

Figure 4.11. JSON file sent to client

4.3 Data gathering

In order to gather data for the purposes of evaluation, additional information such as the number of sensed objects and the volume of data used in the previous communication was parsed into the first JSON file, and sent to the server. There it was stored in tables that described the kind of data that it was. This is discussed further in section 5.1.1.

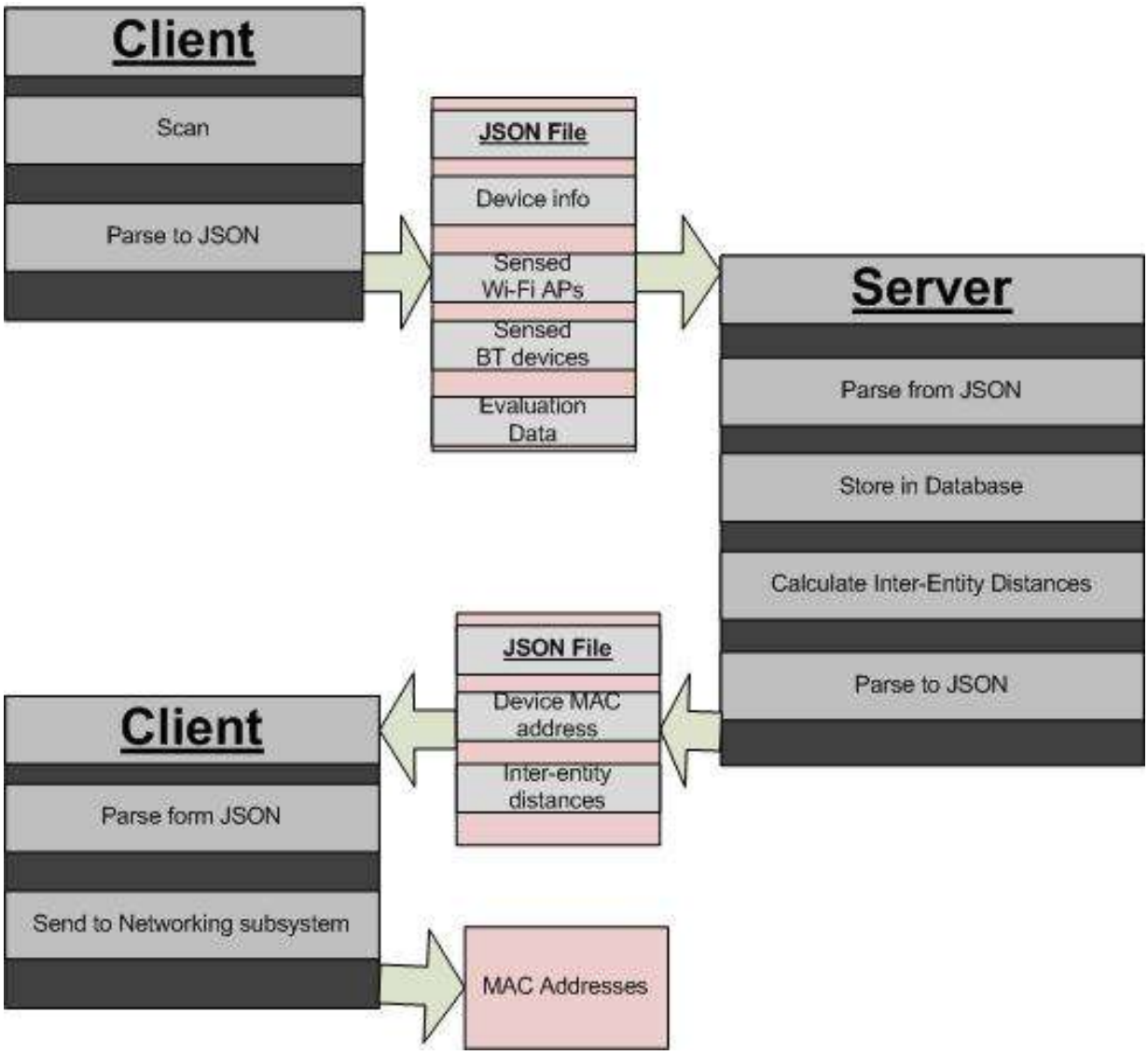

Figure 4.12. Protocol of information passing between client and server

4.3 Summary

The system was implemented in a way that deviated from the system design as little as possible. The decision to implement in Android instead of Windows Phone 7 is the most notable deviation from the intentions set out in the opening chapter. The Design specification was general enough so as to be applicable to wide range of devices, and thus does the decision to move does not affect the research questions, nor expected outcomes of those questions. While unfortunate, the choice to abandon Windows Phone 7 was, at time of this implementation, entirely unavoidable. The remainder of the implementation was successful, with only the cellular tower API incompatibility affecting the calculations of inter-entity measurement.

5. Evaluation

The research question asks whether a new system can be created that discovers devices in the local area, retrieves their Wi-Fi MAC addresses and does so without draining the battery excessively. The sensor radios that allow for device discovery also tend to use large quantities of power. Thus the system developed here attempts to save power by minimising the amount of time that the radios spend scanning. In order to allow for this, the system is designed so that the discovery of nearby devices is performed by a server, rather than the devices themselves. The server relies on devices accurately detecting their own location context by sensing Bluetooth devices and Wi-Fi access points (hereafter referred to as *features*). If this context detection is not accurate enough, then potential networking opportunities may be missed too often to warrant this system's use. The purposes of this chapter are to evaluate how often the system can correctly discover devices within close range, and to evaluate whether the attempt to save power has any significant effect on the device's battery life.

While not being a part of the research question, data-transfer is of concern to users in terms of monetary expense. Thus, data transfer is also evaluated.

5.1 Discovery

The system deals with a potentially very large discovery space (a volume with a radius of up to 500m). It is thus important to remember that the inter-entity distances are merely best-known measurements, and that devices measured as being in the *same neighbourhood* as the client device are not necessarily physically far away from each other. The different measurements of inter-entity distances are merely symbolic ranks of the probability that two entities are in the same physical location. Essentially what the system is describing when it produces an inter-entity measurement of *same neighbourhood* is that the discovered device is at the most, within 1km of the client device, but may also be in the same pants pocket as the client device. Thus, testing the system's ability to correctly measure a large distance between devices does not provide any useful information. The only measurement that is of concern is how often the system correctly measure that devices are physically located close to each other.

Another factor measured in this experiment is how consistently the Wi-Fi and Bluetooth features of each scene are sensed by the scanning function. When executed in the same physical location by the same device, and by different devices, one would expect that each device always senses the same features.

5.1.1 Method

Design & setting

This evaluation is based on an experimental design. It measures the following dependant variables (DV)

1. The inter-entity distance.
2. The number of Bluetooth devices that are sensed.
3. The number of Wi-Fi access points that are sensed.
4. The volume of data transferred during execution (While this data was collected in this experiment, it is analysed in section 5.3).

Comparable results are obtained by manipulating the physical location at which the DVs are measured. Physical locations will be referred to as *scenes* from here on out. Thus scene serves as the independent variable (IV). Six significantly different scenes were chosen from various places around the campus of the University of Cape Town. Scenes were chosen so as to not have any overlapping features between them. They were also chosen to represent a number of different possible scenarios, each differing in the type and number of visible features. The purpose this is to test how critical the occurrence of these features are to the performance of the system.

The term *scan operation* refers to the pressing of the *Scan* button. The term *discovery operation* refers to the pressing of the *Check for friends* button.

For implementation reasons that are discussed in the procedure section below, the scan and discovery operations could not be performed simultaneously, thus there existed the possibility of a sequence effect confounding the data; the possibility that the order of operation would affect the outcomes. Because there were three devices, a simple means by which to control for this is by using *counter balancing*. Counter balancing is a technique that makes sure each possible sequence is tested an equal number of times. The order produced by counter balancing can be seen in Table 5.1.

Materials

The three devices shown in table 4.2 (section 4.1.2) were used for this experiment: The Samsungs Ace and Gio, and the Huawei Ideos. The devices were given Bluetooth names (Jill, Jack and Jess respectively) so as to be identified in a human readable way. It was required that each device be running Android version 2.2 or higher. Each had to have a cellular radio, Bluetooth radio and Wi-Fi radio. Communication with a server is necessary for the system to function correctly. Thus each device was enabled to allow the transfer of data across the cellular networks. Each device was set to the same cellular network provider. Wi-Fi data transfer was disabled control for any extraneous variables created by the switch between Wi-Fi transfer and Cellular network transfer.

Server communication provided a means by which data could be easily and accurately recorded. The client program was configured to send additional data, namely the DVs, to the server, where it was stored in tables in the MySQL database. Table names corresponded to the scene, the device and data that was being recorded. For example, *JessWifiLibrary* holds the records for Wi-Fi features that were sensed by Jess in the Library. This automatic method of data gathering would later save time by allowing the export of comma separated value files from MySQL into Microsoft Excel.

An attempt was made to automate the procedure in order to ensure that the operations for each device performed after an exact time interval of a minute. Android is not a Real Time Operating System, meaning that events and processes to don not run deterministically. Attempts at implementing the procedure with the AlarmManger API kept failing. It was decided that, in order to gather the data without suffering further delays, a compromise would need to be made in terms of accuracy. Thus all operations were performed manually. An Mp3 player with an audio file that played a sound once every minute was used to indicate that a scan and discover operation must be performed. A pen and paper were used to count the number of operations, as well as record some data by hand, so as to identify any possible failed connections between client and server.

Table 5.1

Counter balanced order for each scene

Order	1st	2nd	3rd
Scene 1	Jess	Jack	Jill
Scene 2	Jill	Jack	Jess
Scene 3	Jack	Jill	Jess
Scene 4	Jack	Jess	Jill
Scene 5	Jess	Jill	Jack
Scene 6	Jill	Jess	Jack

Procedure

To collect the data, a number of scan and discovery operations were performed in each scene. The following procedure was followed for every scene. Devices where placed next to each other in the orders seen in table 5.1. The client application was started on each device. The audio timer was started, and placed on repeat. Each device performed a scan operation, in the order dictated by Table 5.1. Immediately following this a discovery operation was then performed on each device in the order dictated by Table 5.3.

The first round of operations results in a *start-up error*. If the order is "Jack; Jill; Jess" the distance measure will be as in Table 5.2. Notice that Jack senses nobody because there are no devices registered on the server yet. Jill goes second and can only sense Jack because Jack has registered, but Jess has not yet done so. Because of this start-up error, the first round of operations were not included in any analysis.

Table 5.2

Example of start-up error

Client	Jack		Jill		Jess	
Entity	Jill	Jess	Jack	Jess	Jack	Jill
Distance	nobody	nobody	Same block	nobody	Same block	Same block

A tally mark was made of for each repetition, and the data value, Wi-Fi number, and Bluetooth number were recorded on paper to double check that all the data was recorded in the database.

This process was repeated every minute, as measured by the audio file, for a period of ten minutes at each scene. Data was collected between the hours of 10pm and 2am. This time period was chosen so that the number of people in each the area was as low as possible. This controlled for any possible extraneous variables that a fluctuating number of people might cause.

Analysis

There is very little existing data against which the gathered data can be compared. Thus the data for this section is analysed using descriptive statistics. In particular, the number of correct inter-entity distances and its correlation to intersection of the number of Wi-Fi and Bluetooth features that are sensed by each device. The variation in the number of Bluetooth and Wi-Fi features that are sensed from the same physical location is also analysed as this affects the way in which physical locations map to relative locations. Finally, the data transfer volumes are analysed in relation to the number of Wi-Fi and Bluetooth features that were sensed.

5.1.2 Results

The first round of results was excluded from the data in each scene, due to the start-up effect (Table 5.2). Given that the devices were always collocated, the system should, ideally, return measurements of *same room* for every round of tests. There are a number of possible factors that could produce results other than *same room*. For example, if the scene does not contain enough features, or if the devices do not sense enough of those features on each scan operation, then the system will not return a *same room* result. Across all scenes, the system detected that devices were within the *same room* range 56.7% of the time; far lower than the desired rate of 100%. The rest of the results can be viewed in Table 5.3 and the distribution of measurements can be seen can be seen graphically in Figure 5.1. A closer inspection of the data is required see why the total number of *same room* measurements only makes up a little more than half of the measurements.

Table 5.3
Summary of the inter-entity distances recorded

	Jack	Jill	Jess	Total	Cum. Total	Average	Percentage
nobody	3	5	7	15	33	5.00	4.39
Same neighbourhood	4	0	3	7	40	2.33	2.05
Same Block	20	21	20	61	101	20.33	17.84
Same Building	17	17	13	47	148	15.67	13.74
Same Room	64	65	65	194	342	64.67	56.73

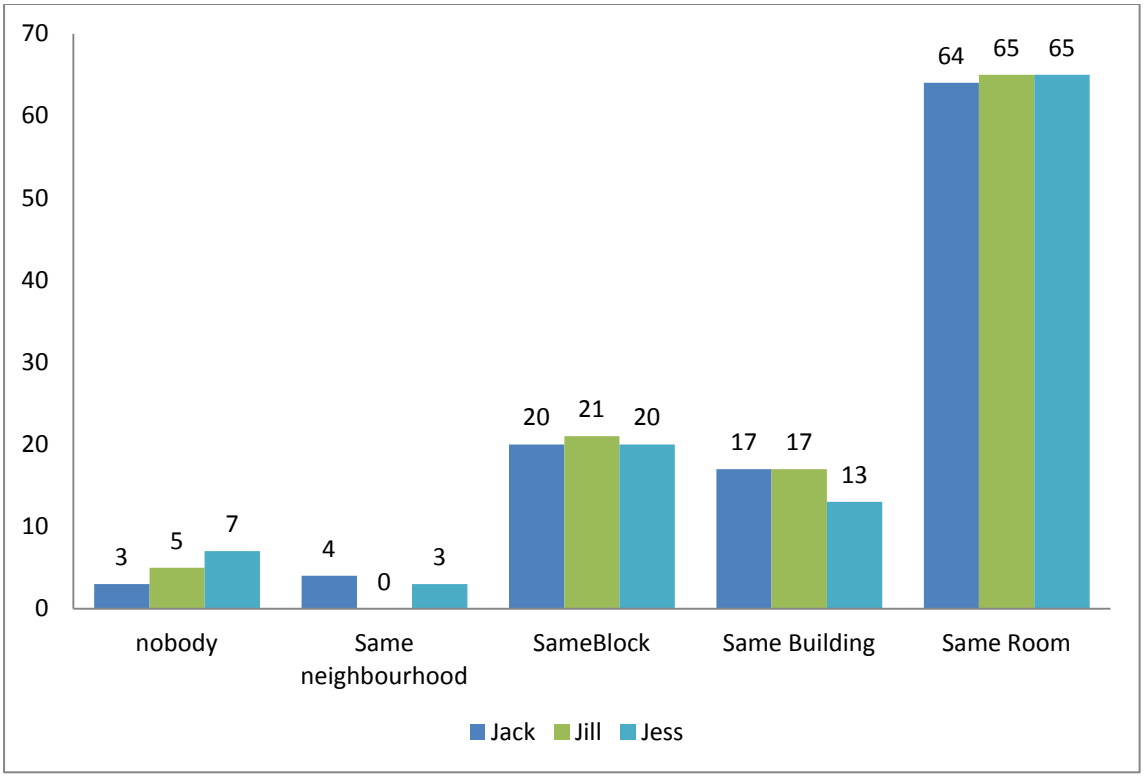

Figure 5.1. Frequency distribution of all measurements taken.

The only scenes that returned *same room* results were those that had at least one Bluetooth feature, as seen in Table 5.4.

Table 5.4
Features sensed means and inter-entity distance summary

	Bluetooth Mean	Wi-Fi Mean	same room distance	other distance
Scene 1	1.13	5.97	51	9
Scene 2	1.17	6.5	56	4
Scene 3	0	0.97	0	60
Scene 4	4.6	6.17	57	3
Scene 5	1.97	10.97	38	22
Scene 6	0	5.27	0	60

An analysis was performed on four of the test scenes. The four scenes were chosen based on the number and type of features that were visible in that scene. The specifics are summarised in Table 5.5. A total of 60 scan and discovery operations were performed at each scene. In the scenes where Bluetooth features were visible, the *same room* calculations number very close to 60. The procedure dictated that each scene was significantly different from the last. Scene 4 returned all measurements as *same room*. Scenes 1 and 2 returned nearly all *same room* results, with just a few *same building*. We have already established the reason for these results compared to the results of Scenes 3 and 6: visible Bluetooth features. Scene 2 is analysed and is representative of Scenes 1 and 4, as the results for all three of these scenes are essentially the same. Scenes 3 and 6 are interesting because they had no Bluetooth features within their area, and Scene 5 is interesting because the *same room* frequency is low despite it having Bluetooth features to work with.

Table 5.5

IV differences at each analysed scene

Scene 2	Small number of Bluetooth features sensed; More than one Wi-Fi feature sensed
Scene 3	No Bluetooth features sensed; roughly one Wi-Fi feature sensed
Scene 5	Small number of Bluetooth features sensed; larger number of Wi-Fi features sensed
Scene 6	No Bluetooth features sensed; More than one Wi-Fi feature sensed

Scene 2

This scene produced a set of results that indicates the system works well; it was able to measure that the collocated devices were in the same room for the majority of the test, with only one other inter-entity distance measurement that is not *same room* (Figure 5.2).

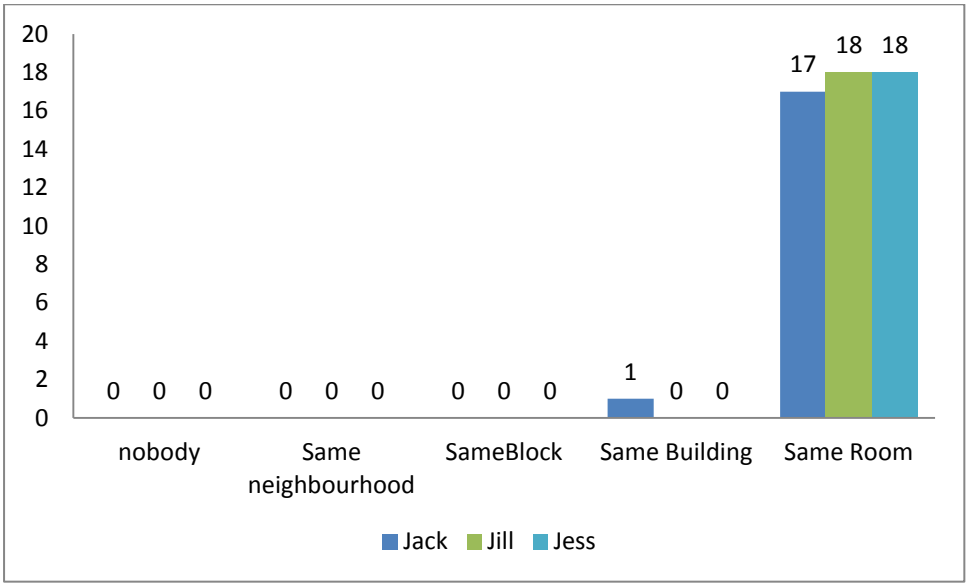

Figure 5.2. Distribution of inter-entity distance estimates at scene 2.

The Bluetooth feature results show that there where at least two devices present (Figure 5.3). The mean of 1.16 Bluetooth features with a standard deviation of 0.37 do little but show that the number of Bluetooth features sensed was consistent. The raw data shows that Jack did not share Jill's Bluetooth feature during test 9, resulting in the *same building* estimate. However Figure 5.3 shows that they both did sense a Bluetooth feature in test 9, just not the same one. This means that while the two devices share an almost identical physical position, their relative scene is not the same; in other words, their discovery spaces do not intersect as well as they should, given their collocation.

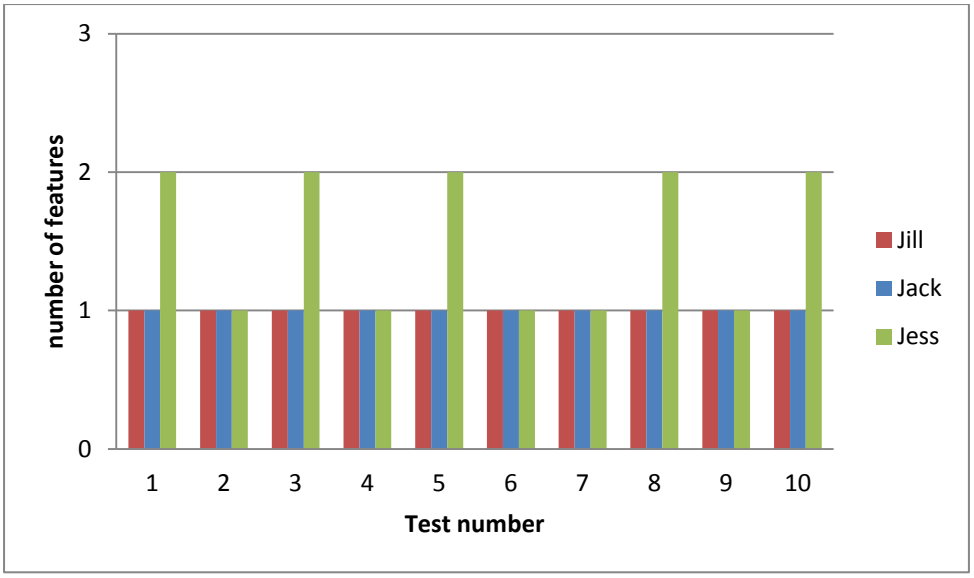

Figure 5.3. Overview of Bluetooth features sensed for test at scene 2.

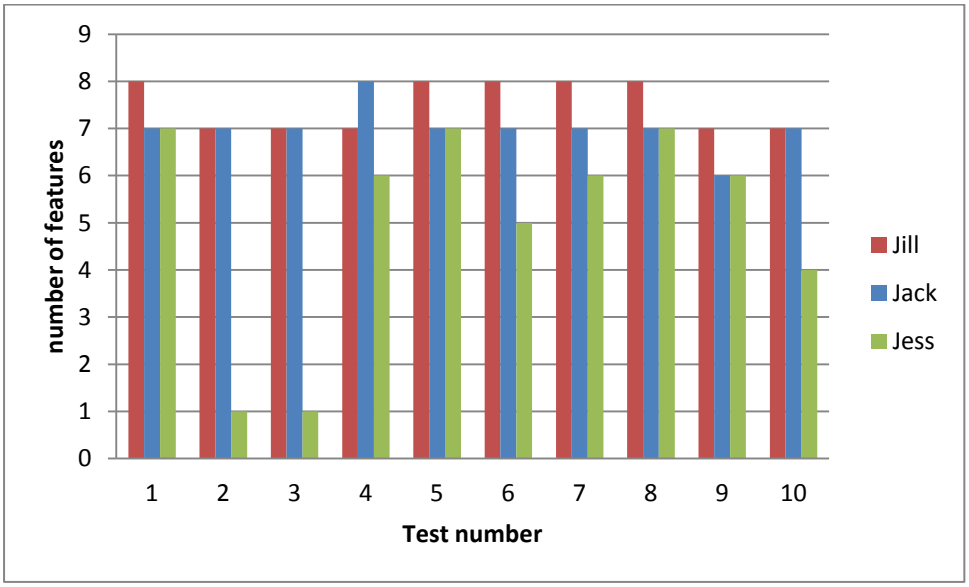

Figure 5.4. Overview of Wi-Fi features sensed for test at scene 2.

The Wi-Fi feature overview seen in Figure 5.4 presents similar data. The mean number of Wi-Fi features sensed is 6.5 with a standard deviation of 1.7. The maximum number of features sensed is 8, and the minimum is 1. This indicates that the number of features that each device is sensing is consistently high and is fairly stable. Jess' mean is 5, and it has standard deviation of 2.3. Jess' has a maximum of 7 and a minimum of 1. These wide ranging values suggest that, in this scene, Jess's sensing abilities are possibly less consistent than the others'.

Scene 3

Scene 3 contained no Bluetooth features, and contained two sense-able Wi-Fi features. Each device had a mode of one Wi-Fi feature, indicating that the second feature had a very weak signal. Jess failed to sense either of the Wi-Fi features on three occasions; tests 2, 9 and 10 (Figure 5.5). There occurred more *nobody* measurements, as seen in Figure 5.6, than failures, on Jess' part, to sense features. This must mean that while each device sensed a Wi-Fi feature, it was not necessarily the same feature (example, Jack and Jill sensed Wi-Fi feature A only, while Jess sensed Wi-Fi feature B only). The results in this scene are suggest that the number features which are sensed in a given scan can be substantially different to the features that are visible in a given scene. These discrepancies become more apparent in the analyses of the subsequent scenes.

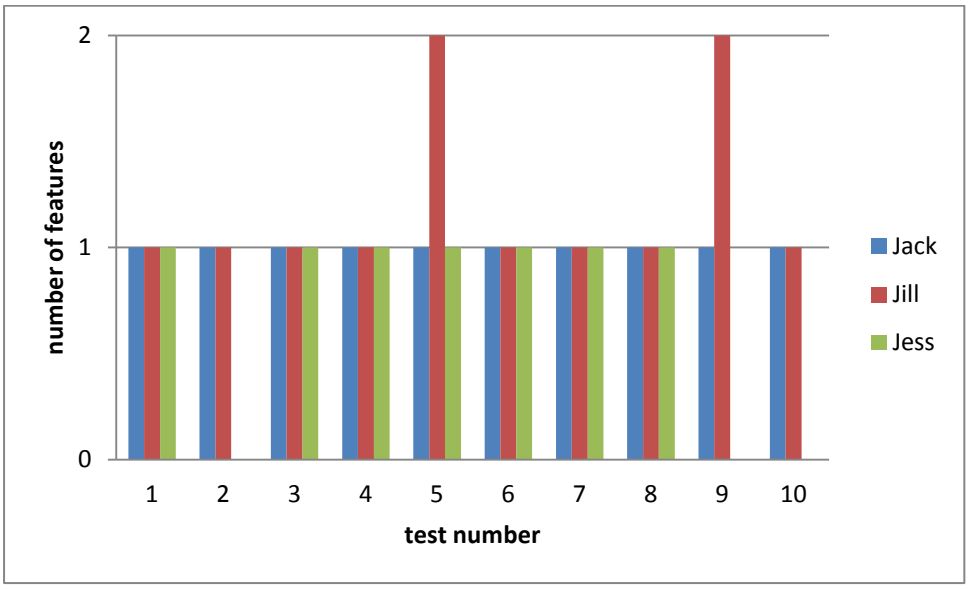

Figure 5.5. Overview of Wi-Fi features sensed at scene 3.

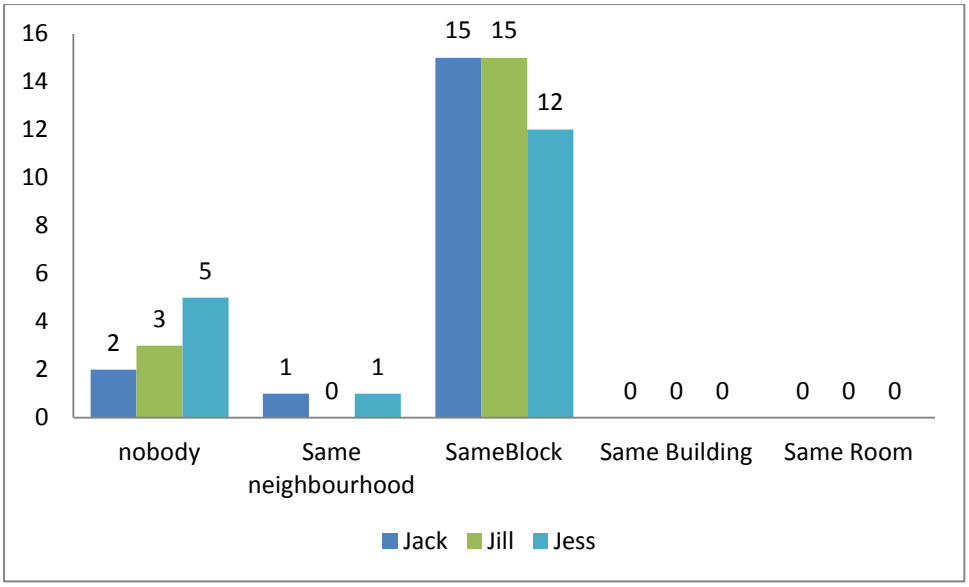

Figure 5.6. Distribution of inter-entity distance estimates at scene 3.

The fact that there are so few *same neighbourhood* measurements indicate that more than one Cellular tower was in range during the tests, and it is likely that Jill did not share a Cellular tower with the other two devices. If the devices had been able to sense more than one Cellular tower with each scan, there may have been more inter-entity distance measurements indicating that the devices were in the same discovery space (same neighbourhood).

Scene 5

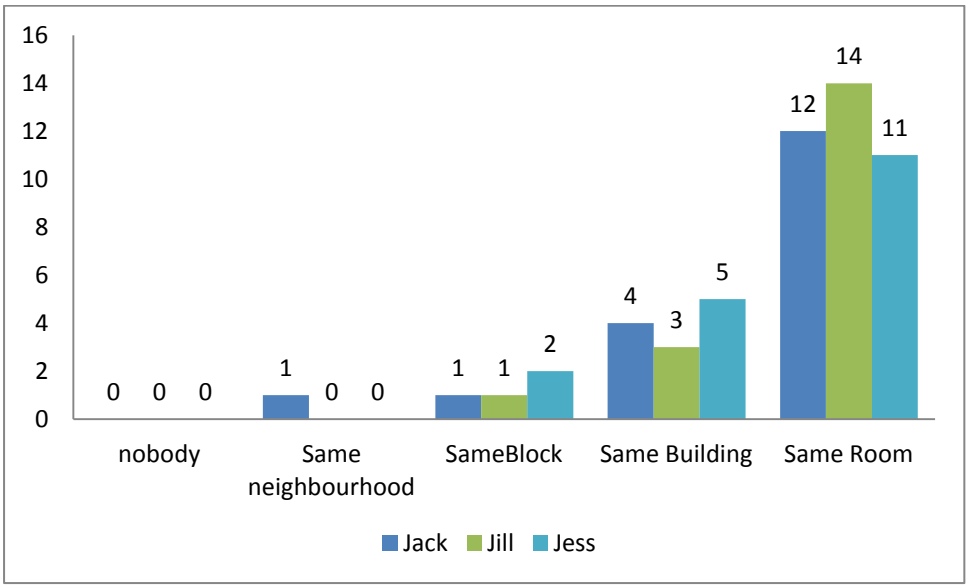

Figure 5.7. Distribution of inter-entity distance estimates at scene 5

Scene 5 included Bluetooth features, but did not have the same high numbers of *same room* measurements that the other scenes that included Bluetooth features. This as can be seen in Figure 5.7. This poses questions about why it should be different from scenes 1, 2 and 4 (Table 5.5). To answer them we again look at the test overviews of Bluetooth and Wi-Fi features.

The mean number of Bluetooth features sensed is 1.96, with a standard deviation of 1.1. This implies that every device was sensing at least one Bluetooth feature on every test, but was not sensing all of them. However, if we examine the overview shown in Figure 5.8, we see that while Jack and Jill have Bluetooth feature modes of 2 and 3 respectively; Jess most commonly sensed no Bluetooth features at all. In particular, on test number 8 Jill sensed 3 Bluetooth and Jack sensed 4 while Jess sensed 0. That is a large discrepancy when considering that all devices were in the same physical location while performing scans, and implies that scanning for features does not return consistent results.

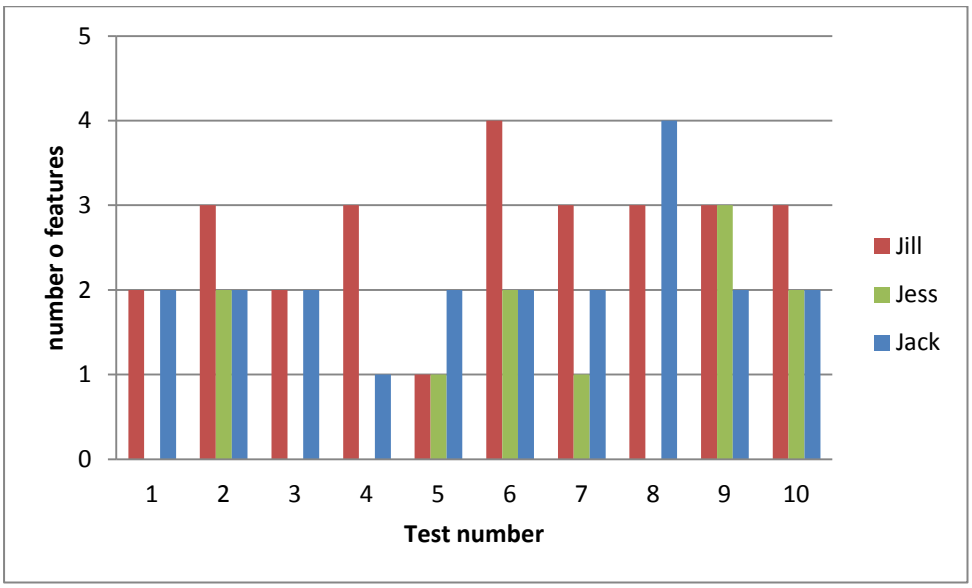

Figure 5.8. Overview of Bluetooth features sensed for test at scene 5.

The Wi-Fi overview shows similar results (Figure 5.9). The mean number of Wi-Fi features sensed is 10.96 with a standard deviation of 4.2. Again, considering that all devices were in the same physical location, it was assumed that the number of features sensed would be constant. However, these figures indicate otherwise. One cannot attribute the standard deviation to differences in device set up either. Jess sensed the following number of Wi-Fi features in three consecutive scans: 0, 15 and 1

(Figure 5.9). Jill has a mode of 14 for scene 5, and sensed this numbered for six out of ten scans. However, in the remaining four tests Jill sensed as few as 5 Wi-Fi features, and as many as 18.

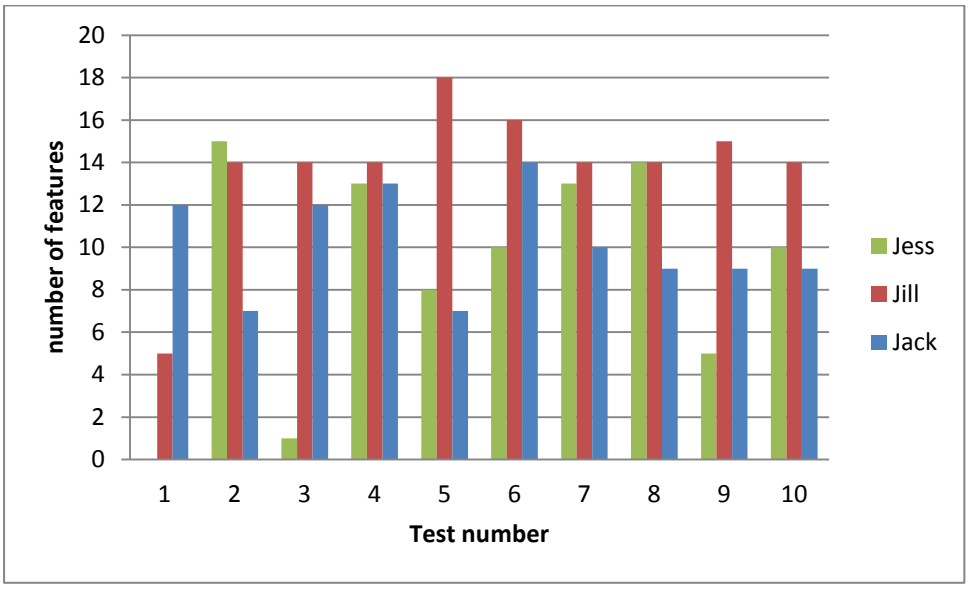

Figure 5.9. Overview of Wi-Fi features sensed for test at scene 5.

These results further suggest that the features available to a device in one physical position are not necessarily constant.

Scene 6

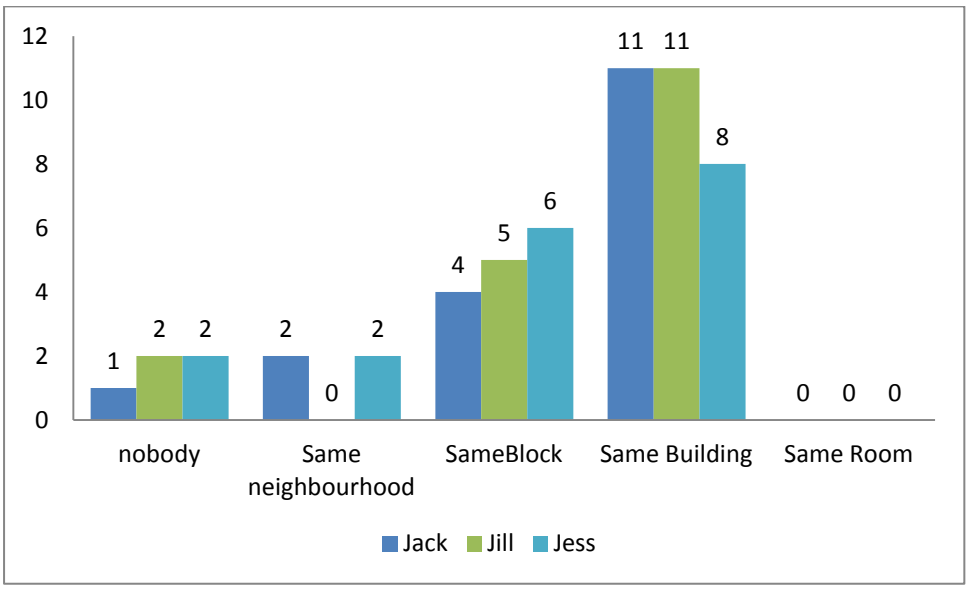

Figure 5.10. Distribution of inter-entity distance estimates at scene 6

This scene did not have any Bluetooth features, but it did have significantly more Wi-Fi features than were visible in scene 3. Thus there are more *same building* results than in scene 3 (Figure 5.10).

The mean number of Wi-Fi features sensed by all devices is 5.2 and there is a standard deviation of 2.1. With these statistics one would expect that the number of *same building* measurements would be higher, because the number of features that are sensed by each device is greater than one; even when the standard deviation is subtracted from the mean. Figure 5.10 shows that the maximum number of features in one scan is 9, and the minimum number is 0. Jess is responsible for recording the 0 value on two occasions. Every other value is two or above. In fact there is only one recorded value 'two' on for this scene. The server will only post an inter-entity distance of *same block* if there is exactly one Wi-Fi feature in common, and it did so 15 times. This accounts for a quarter of the discovery operations performed in this scene. Evidently, while the number of sensed features is high enough to produce *same building measurements*, the number of *shared* features is not. For example, the raw data shows that Jack and Jess have an inter-entity distance of *same block* on test two, and yet both Jack and Jess registered 5 Wi-Fi features on this test. We have established that the maximum recorded number of features is 9. It would seem that

union of Jack and Jess make up this value of 9, because the value their intersection is 1.

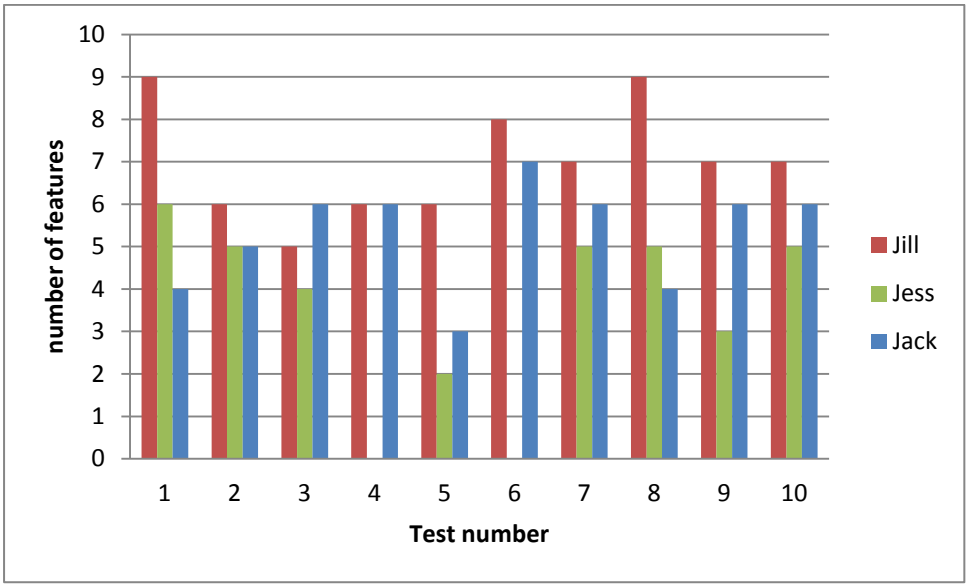

Figure 5.11. Overview of Wi-Fi features sensed for test at scene 6.

5.1.3 Discussion

Summary and interpretation

The system appears to function well in the presence of Bluetooth features. Given this data, if one were to remove the two scenes in which no Bluetooth features were available, the system correctly estimates that the devices are in the *same room* 84.2% of the time. However, this rate potentially drops to zero when the Bluetooth features are removed. This result is not unexpected, as the system is not designed in a way that makes it easy for a distance of *same room* to be measured without the aid of Bluetooth. A closer evaluation of the system when running without Bluetooth features reveals a dynamic that is of concern. The scan operation does not sense all the features of a particular scene with much consistency. The results described above show that while the number of features sensed by each device can be high, the intersection of these features (the number that decides the inter-entity distance) can remain low. Thus the system's accuracy at sensing collocated devices can be called into question.

Limitations

The reasons for the lower than expected results, particularly in scene 5, may be caused by do with fluctuating signal strengths. Bahl & Padmanabhan(2000) described how Wi-Fi signal strength can change by up to 5dBm (measure of signal strength) depending solely on the direction in which the client device is facing. HP (2002) explain that Bluetooth and Wi-Fi both share the same section of the 2.4GHz ISM band of radio space. The two radios could interfere with each other during scanning. With this in mind, the internal validity of the experiment is called into question. The procedure dictates that once a minute, each device must perform a scan. The process is staggered slightly, because the experimenter could not operate all devices simultaneously (See 5.1.1: Materials); however the scans do all execute within the same minute possibly causing interference. If the process were to be changed so that the scan operations were staggered over a three minute period, with one device performing a scan every minute, and then performing a discovery operation on the fourth minute, the results may prove to be different. A useful measurement to add to this experiment would be the MAC address of the features that are sensed by each scan operation, providing insight into which features are shared between devices.

Conclusion

In conclusion, the results show that the system performs well when Bluetooth features are present (scenes 2 and 5). They also show that the system relies heavily on network infrastructure in order to perform accurate measurements (scene 3). Furthermore, it shows that even in the presence of established infrastructure, results can be inaccurate due to the inconsistent sensing of the features in a particular scene (scenes 5 and 6). As this system is designed for Android, and makes use of a Bluetooth radio, one possible means of increasing the success rate is by making the client device discoverable when a scanning operation is performed. Throughout the results section, Jess appeared to be the only device that tended fail at sensing the available features. For two of the scenes analysed, Jess was last in order of operation, and was second and first for the remaining two scenes. If the scanning operations are indeed interfering with each other, then Jess might have failed to sense these features due to the interference from the other two devices. However, Jess also failed on occasion when it was first in order, namely Scene 6. It could be that the hardware

in the Huawei is less sensitive to broadcasts from Wi-Fi and Bluetooth features than the hardware in the Samsungs. Results from the Battery experiments suggest that there are differences in hardware between the two Samsungs as well. This suggests that hardware, and not just scene infrastructure, play a crucial role in the establishment of discovery space.

5.2 Battery

Section 2.1.1 indicates that Wi-Fi is one of the subsystems that consume the most power, particularly when scanning for access points. Thus we hypothesise that the scan operation would consume more power than the discovery operation. The GROUT device discovery system was built based on this assumption; it attempts to minimize the amount of time spent scanning.

Measurements of battery consumption were required to test our hypothesis. However, finding ways in which to measure how much battery power is used by the subsystems was not a simple matter. Battery profiling tools do exist, but they tend to be device specific (PowerTutor, 2009; Intel, 2011). Both PowerTutor and Intel's Power Profiler for Android were tested on all three devices, and while the software did run on the devices, the details needed to successfully estimate the power consumption of the radios were either missing or incorrect (See appendix C).

In general, unless specific profiling tools exist, monitoring the power consumption of devices is difficult and can required that third party hardware be used to take measurements. Rice & Hay (2010) developed such a system to measure the battery use of Android devices. They built battery replacement devices; systems that produced performances that were indistinguishable from the standard batteries associate with those devices. The systems contained hardware, such as resistors and a power sampling board, which allowed accurate measures of power consumption. The authors claim that this system is available to normal developers, in the sense they need not perform any serious hardware modification. Awareness of this system was only gained post evaluation, and it use was not employed

The sections that follow include the methods of gathering battery information that were explored. Experiments were not carried out in all cases, thus there exists no data for those cases.

5.2.1 Native methods for reading Battery data

The android devices used in this evaluation all have a “battery usage” feature. This feature estimates the percentage power devoted to specific hardware subsystems, such as the Bluetooth, and Wi-Fi radios. To calculate these values, the Android OS uses what are known as the *hidden* and *private* APIs. Inazaruk (2011) explains that these are APIs that are not accessible to developers using the standard SDK, because they are not accessible at compilation time. They are however used be the operation system at runtime at. These limitations are imposed by the SDK and the Eclipse IDE and can be circumvented by importing library containing these APIs into an Android project. Using this it appears that detailed readings of battery usage are accessible. Research failed to identify official documentation relating to these API. Grepcode (2012) provide a means by which the API can be browsed, but there is little explanation. The code comments on, and briefly explains some of methods and variables, but is also littered with TO DO comments; large sections of incomplete code. Using these documents as a guide, what was described as the average milliampere used by each subsystem was obtained. In addition, the amount of time for which the subsystem was in use could be calculated; something not possible using the regular API. Unfortunately, the methods that were meant to return a value representing when the subsystems were in a high powered state consistently returned a value of zero. Thus the only readings of battery use obtained appeared to be the average current used by each subsystem while switched on, and nothing more. The GROUT client application is designed in such a way that the Bluetooth and Wi-Fi radios remained switched on while the application is running. Without a reading that differentiates between the current used in the subsystem’s *on* state and in the subsystem’s *scanning* state, this method of evaluating the battery is not useful as it will produce the same result for both the scan operation and the discovery operation.

5.2.2 ‘Top’ command method

The Android OS is a software stack based on the Linux operating system. Linux has a shell command *top* which looks at each process currently running on the system, and measures the percentage of the CPU that is being used for that process. Measurements are taken at regular, user specified intervals. *Top* prints out a list, ranking processes in order of the percentage CPU that the process is using. This list will henceforth be known as the *top list*. The Android Debug Bridge (adb) is a

command line tool that allows shell commands to be run on instances of devices, or emulators. *Top* was chosen for use with the hope that by observing the CPU usage, there would be an indication of when the radios were put to use and for how long they are in use. A short experiment was run to access the feasibility of using this command for evaluating battery.

Design & setting

Based on an experimental design, the test sought to measure the CPU usage of the client application, Wi-Fi and Bluetooth when a scan operations were completed and when discovery operations were completed. We define CPU usage as the dependant variable (DV) and the type of operation as the independent variable. The experiment takes place at a suitable terminal in a location from where a connection to the server can be established.

Materials

A computer that has the SDK installed on it is required in order to run adb, which must be executed through a command line interface. The client application can be run on the emulator, but the experiment also must measure Bluetooth and Wi-Fi and hence requires an android device, and a USB connector cable through which the computer and device can interface. A stop watch is required to time the intervals between operations.

Procedure

A command line tool was started and the directory containing *adb.exe* was navigated to. The device on which the experiment was to be run was connected to the computer. In this case it was the Samsung galaxy Ace (Table 4.1). The following procedure was followed for both experimental conditions: All applications on the device were closed and memory was cleared from the Android task manager before the client application was started. The top command was set up to execute once every second, and print the top 5 results of each execution to a text file on the computer. The stopwatch, top command and operations were begun at roughly the same time. This being a feasibility test, complete accuracy was not required. Each scan was run for approximately The scan operation was run first, additional scans were performed as soon as the *scanning* text on the device changed to a zero, indicating that the scanning was complete, and the time on the stopwatch at this point was recorded.

For the discovery operation, the *Check for friends* button was pressed only at the times recorded in the *scan* test. The results where then transcribed into an excel file.

Analysis

No statistical analysis was used here. All conclusions were drawn from observation alone.

Results

The results for the *scan* and *discovery* runs can be seen in Figures 5.12 and 5.13 respectively. In the *scan* test, scans were performed at around time 1, 13, 27, 39 and 50. As can be seen in Figure 5.12, these times roughly correspond to the points were the Client application registers on the *top* list (red x's), as well as the times that Bluetooth registers on the *top* list. The Client never ranks higher than number 4 on the list in this case; there were always process using more of the CPU than the client app. The results from the *discovery* test displayed more CPU usage for the Client app and less usage from the radios. CPU usage outside of the times when the button was pressed might indicate a data returning from the server. In this test, the client ranked much higher more often than in the previous test. However, there is not much detail in these results; a result that may be due to the latency between *top* iterations. One second is a long time for computers. Additional tests were performed with smaller intervals between *top* iterations, but the *top* process itself very quickly grew to using 100% CPU power, and the other detail was lost. Due to the lack of detail, the strategy to monitor the battery usage through the use of *top* was abandoned.

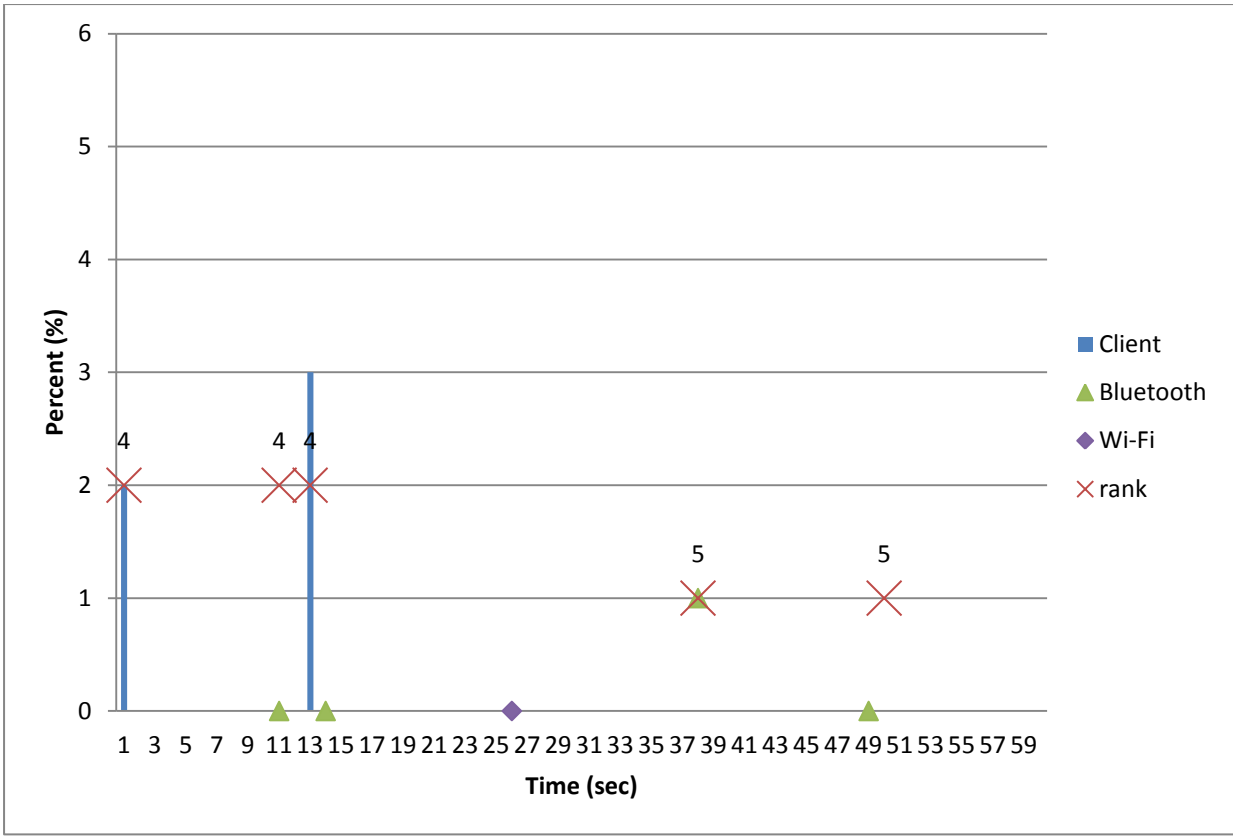

Figure 5.12. Results of the scan operation from the top command. The red X's represent the Client app only, and shows each time it registered in top's output. The number above the x indicates the rank on the list. The blue columns show the percentage of the CPU that was being used by the client app at that time (no column corresponds to CPU usage of 0%). The Bluetooth and Wi-Fi points show their appearance on the top 5 list, as well as their CPU percentage.

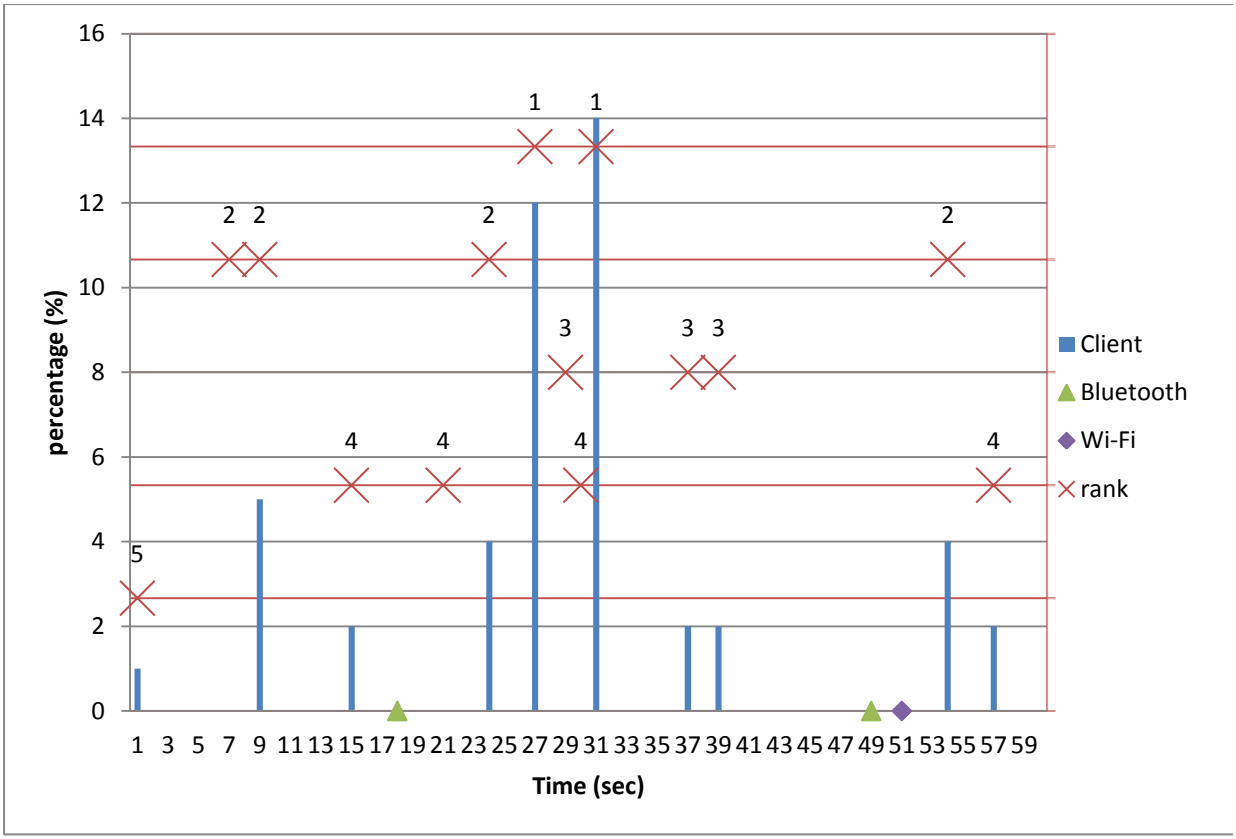

Figure 5.13 Graph showing the results for the discovery test.

Discussion and conclusion

The results seen in Figures 5.12 and 5.13 suggest a correspondence between CPU usage and the *scan* and *discovery* operations, but this is to be expected. The system is designed to use the radios for *scan* operations, and to not use them during *discovery* operations. The rarity with which the radios appear in the *top* list indicate that even if they are active, at least five other process are using more of the CPU than they are. The client itself only ranked number one on the *top* list twice.

Top did not provide information regarding the battery use in the GROUT scanning system that was detailed enough to warrant further testing. The results did hint that other processes often use more of the CPU than the client, or the radios.

5.2.3 Battery percentage monitoring

The Android devices used in the evaluation display the amount of battery remaining to the user with a graphic and as a percentage. Neither may be specific, but the latter can be used to measure battery consumption over periods of time. In an experiment it can be used to estimate how much each test of that experiment uses, and the relative differences can be used to test whether or not the one uses more battery than the other.

Design & setting

This experiment utilised a within-subjects design to ascertain how much battery is used by each independent variable a period of time. The independent variables (IVs) are the *scan* and *discovery* operations, and the dependent variable (DVs) is the amount of battery left after a timed period of use. The same test is repeated for both IVs on the same device. Doing so controls for the hardware differences between two devices. Other extraneous variables for this experiment include any function that may use battery power for anything other than the performance of the IVs. This can include uncontrolled scanning for cellular towers and Wi-Fi access points when not performing a *scan* operation.

Materials

This experiment requires at least one device. The results might be more valid if they are confirmed to be relatively consistent on two different devices. Hence, a Samsung Galaxy Ace, and a Samsung Galaxy Gio where both used in this experiment. Other materials include a charger in order to re-charge the devices before performing the experimental tests. A means of keeping track of time was required in order to regulate when to perform an operation, as well as when to cease the current run of the experiment. An alarm was used for both, one that repeated every minute, and one that went off after an hour.

Procedure

The experiment was repeated a number of times on each device. The initial percentage charge of each device's battery was recorded. The device was then left to discharge. After one hour, the percentage charge of each device's battery was recorded again. The difference was calculated between initial reading, and end reading. This was done to ascertain the *Standby* battery usage, which would serve as

a baseline percentage against which the other two operation's battery usage could be compared. The devices' initial charge was not kept constant. This discharge process was repeated six times over to gain a more representative average of the baseline discharge. Once the baseline was established, the experiment was run another twelve times. This yielded six discharges per operation. Larger sample sizes were not collected due to time limitations. For all iterations, the screen timeout was set to two minutes so that the devices did not lock their screens in the time between operations, and then use additional power in order to switch the screen on and off. The devices were fully charged prior to each discharge. The initial charge was recorded, the device was taken off charge and then a *scan* operation was performed every minute for a period of one hour. At the end of this period, the charge was measured again, and the difference calculated. The devices were fully recharged before each of the iterations. The initial charge was recorded, and a *discovery* operation was performed every minute for the period of one hour. At the end of each hour, the charge was recorded and the difference calculated.

Analysis

Descriptive stats were performed on the data. A paired samples, one-tailed t-test was then performed on the data test if scan operations used significantly more battery than discovery operations.

Results

The standby tests recorded a mean battery use of 4.03% and 6.08% was recorded for the Gio and the Ace respectively. The Ace uses slightly more battery in its standby state than the Gio, but differences were to be expected due to the devices being different models, and running different versions. However, the means for the operations showed significant differences in power usage. They differed by over 50 percentages between devices in both the operation tests, as can be seen in Figure 5.14. This oddity is discussed in the Limitations section below.

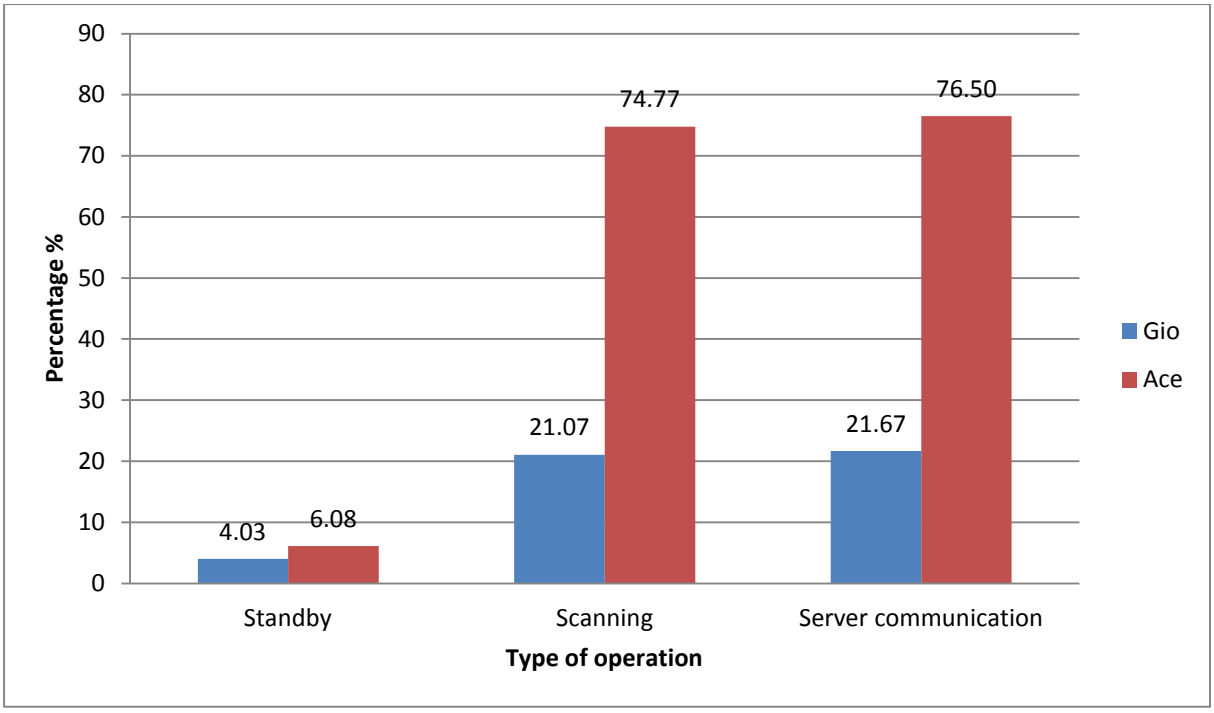

Figure 5.14. Graph showing the percentage of battery used in the three tests.

The results comparing the standby tests to the scanning operation are summarized in Table 5.6. The difference between the percentages of battery used in each of these operations is significant at the 0.1 level; indeed they are significantly different at the 0.01 level. The results for the test between Standby and discovery operation is not discussed as the results are effectively the same as these. From these results we can be fairly certain that each operation as performed in this experiment uses more power than when the device is in standby mode.

Table 5.6

Descriptive and inferential statistics between standby and scan tests

Device	Gio		Ace	
Operation	Standby	Scan	Standby	Scan
Mean	4.03	21.07	6.08	74.77
Standard deviation	0.52	2.72	2.21	2.53
P Values	P<0.001		P<0.001	
Sample size	6			

The results of the inter-operation t-tests are summarised in Table 5.7. The difference between scan and discover operations was not significant for either devices at the 0.01, 0.05 and 0.1 levels. This indicates that the hypothesis must be rejected. The

scan operation mean for both devices are lower than their respective discovery operation, however each mean falls within the standard deviation of the other operation's mean. This shows that results as seen in the discovery operation can easily be seen in the scan operation, and vice versa.

Table 5.7

Descriptive and inferential statistics between scan and discover tests

Device	Gio		Ace	
Operation	Scan	Discover	Scan	Discover
Mean	21.07	21.67	74.77	76.50
Standard deviation	2.72	4.43	2.53	3.30
P Values	P > 0.1		P > 0.1	
Sample size	6			

Limitations

When performing t-tests, assumptions are made regarding the sample.

1. The samples used come from a population that is normally distributed
2. The samples are independent of each other
3. The variances within each set of samples are homogenous.

It is not known whether the population from which the samples were taken is normally distributed. The sample size is not large enough to check for normality. This may cause the results to be invalid. The Ace has a mean that is roughly 55 percentages higher than the Gio's mean. The Ace's standard deviation was 2.53 for scan operations and 3.3 for discovery operations. The high mean was therefore not the result of outliers. The battery usage feature (mentioned in section 5.2.1) revealed that the display was responsible for the majority of the battery usage. A test was performed where the Ace was fully charged, and then allowed to discharge with the display switched on. In the period of an hour, the display drained 75% of the battery. The same test performed on the Gio resulted in 10% of the battery being used. This is a notable difference in subsystem consumption of devices, and therefore display power usage is considered to be a confounding variable within the test, possibly affecting the validity of the results.

Discussion and conclusion

The Battery percentage was never going to be an accurate measure of radio use. However, by running a series of tests using each operation, it was hoped that a relative difference would be found between the scan operation and the discovery operation, from which it could be inferred that the scan operation uses more power. The results, however, showed no significant difference between the two operations. These results may not be valid, due to the assumption of normality being violated. The display subsystem on the Samsung Ace turned out to be incredibly power demanding and may have confounded the results for that device. This was not the case with the Gio, however, as it used considerably less battery to power the display; am mere 10% of the battery was used in one hour of constant display. This amount is only 6% more than the standby power usage. These figures highlight the differences in power usage between devices. The results of the experiment performed in section 5.2.2 hinted showed that other processes were using more CPU power than any process involved in the GROUT device discovery client application. The results in this section show that this is may be the case with other subsystems too; as is the case of the Ace. The display alone will drain the battery in under two hours even if no other subsystem is used. Thus it is concluded that in some devices, for example this Samsung Ace, battery usage by the radios should not be the focus of concern with regards to power consumption.

5.2.4 Conclusion

The evaluation of the battery usage provided no conclusive results regarding whether or not the GROUT device discovery system saves power with its design of performing fewer scans. Difficulties in data gathering, and the presence of extraneous variables all contributed to this. However, the results of the experiments are not entirely fruitless, highlighting other problems associated with power consumption of mobile devices; some not considered in previous sections. The CPU readings obtained in section 5.2.2 and the extremely high cost of powering the display for the Samsung Ace used in these experiments show that while the power use of the radios is of concern in context of scanning and transfer alone, they can be inconsequential in relation to the power use of the rest of the device. Nonetheless, more research is required to test the battery usage properly, identify if the hypothesis should indeed be rejected, or if it can in fact be accepted.

5.3 Data transfer

Keeping the amount of data used to a minimum has been identified as being crucial to the utility of a device, particularly when there is monetary cost involved. If the data cost of the system is too high, it will not be useful as a device discovery method. Thus the amount of data transferred for every discovery operation was recorded in order to measure how the cost of data transfers might change as the number of features in a scene changes.

5.3.1 Method

Design & setting

The data for this was collected simultaneously with the data for section 5.1. Thus this section is identical.

Materials

Data was calculated through the Android *TrafficStats* API. It allows the calculation of the total number of bytes sent and received through the all network interfaces. By switching off all background data usage, and working out the difference in the number of bytes sent and received between discovery operations one can get an estimate of the amount of data required to send and receive the information between client and server. More accurate methods in the API could not be used due to API level 8 restrictions.

Procedure

The data for this was collected alongside the data for section 5.1. Thus this section is identical.

Analysis

Because the data value is the difference between two separate discovery operations, the first reading is always zero. These values were not considered in the analysis.

5.3.2 Results

Global

Table 5.8

Device data usage in bytes

	Jack	Jill	Jess
Mean	4665.33	1874.796	2006.5
Mode	1913	1826	1757
Variation	19011668.7	72045.07	81036.22
Stdev	4360.24	268.41	284.67

Jack's mean is substantially higher than that of Jill's and Jess's. The same pattern is seen in the standard deviation. Despite closing background data transfer, there is some data transfer occurring that is not related the GROUT system. Figure 5.15 supports this by displaying a negative data transfer reading, which indicates that Jack received more data than was sent in the previous test. The JSON file sizes are comparable those sent and received by the other devices, whose data transfer sizes remain consistent. The results from all tests indicate that these extra data transfers are common for Jack, but not do not occur all the time. Thus we can conclude that Jack is sometimes affected by extraneous variables. Nonetheless, Jack is left out of the analysis.

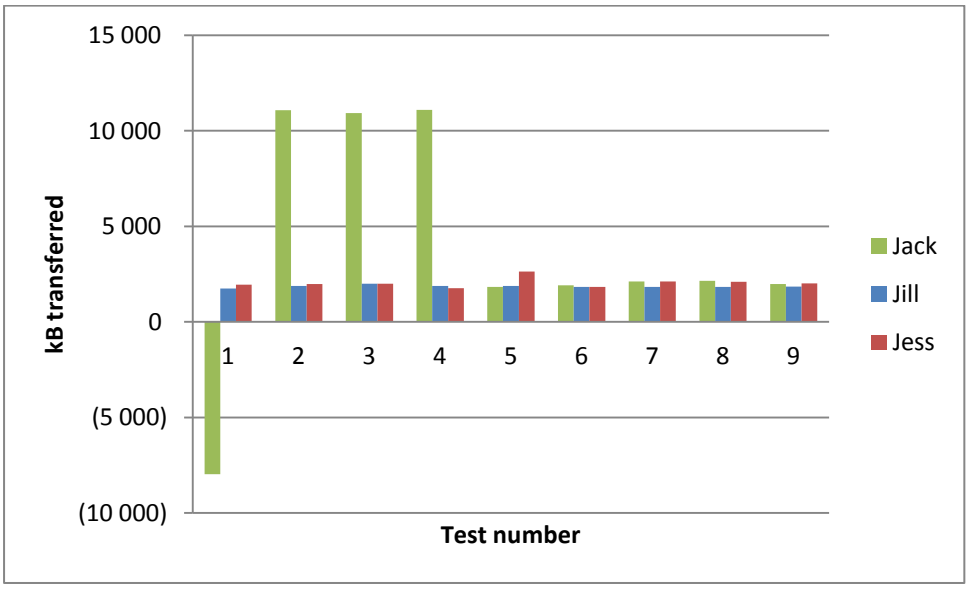

Figure 5.15. Data transfer volume overview for scene 1.

The descriptive statistics for Jill and Jess are similar. The Mean is relatively small there is a small standard deviation. We would like to see if there is a relation between the data size and the number of features sensed in each scene.

Using all the data for Jill and for Jess, correlation coefficients were calculated for the relationship of data transferred and the number of features in a test. For Jill, a correlation coefficient of 0.67 was calculated. For Jess, a correlation coefficient of 0.29 was calculated.

5.3.3 Discussion

Summary and interpretation

The amount of data transferred with each discovery operation was evaluated. Extraneous variables meant that all data for Jack was disregarded. Data for the other two devices were analysed. Correlation coefficients were calculated for both Jill and Jess with regards to number of features and amount of data transferred per test. The results show that both devices exhibit a positive relationship between the number of features and amount of data transferred. For Jill, the result indicated that there is a substantial relationship between number of features and amount of data transferred, and that the relationship moderately positive. For Jess the coefficient indicates that there is a relationship, but it is small, and the correlation between number of features in a scene and the amount of data transferred is a weak, but positive one. The size of the JSON file will of course have an effect on the amount of data transferred, but this result indicates that it might not be much, in the case of Jess, or it might be a moderate amount in the case of Jill.

Limitations

The large mean and standard deviation associated with Jack indicates that the method used for collecting this data is not necessarily accurate. Any data not associated with the client application that the devices transferred during testing will affect the data reading.

Conclusion

From this result we can conclude that when the number of features in a scene is higher, then the amount of data transferred does increase. The results produced by Jill indicated that the amount of data being transferred was closely linked to the size

of the file. This is good because, the actual amounts of data were relatively small. MTN (Date unknown) charge per kilobyte, and given the mean of Jill, these sizes are about as low as can be. Jess's results showed a smaller correlation between the file size and volume of data. This suggests that some of the data being used was unrelated to the GROUT system, and was being transferred regardless of whether the system was being used. Jack's results were discarded for this same reason, and both were transferring data despite the fact that the background data transfers had been switched off. This indicates that there are processes that consuming small amounts data executing constantly, which over time adds up to more than any amount of data used by the GROUT system alone.

6. Conclusion

6.1 Summary

In summary, this was a report on research on and implementation of a system with which mobile devices can become aware of other mobile devices in the same vicinity as themselves, so that they might form ad-hoc networks that do not rely on networking infrastructure. While the intention was to implement this system on the Windows Phone 7 operating system, but the limitations of this platform dictated that it was not possible to do so at the time of implementation. Thus the system was developed for the Android operating system.

The system was designed and implemented with the following research questions in mind:

1. Can a device discovery system be created that successfully harvests the MAC addresses of all collocated devices while saving power?
2. Can this device discovery system perform its function without transferring unreasonable amounts of data?

The system was then evaluated in a way that might answer the research question explicitly.

6.2 Discussion of research questions

6.1.1 Question 1

In order answer this question with a yes, the GROUT device discovery system was designed so that discovery could be performed without requiring scan from the power consuming radios. A literature survey provided examples of systems that suggested performing the actual discovery with on server; all the mobile devices had to do was perform one scan in order to register their relative location on the server. The theory behind this is that after one scan, a device can perform any number of discoveries without requiring additional scans, thus not using the power hungry radios and therefore saving battery power.

It was found that client devices using the system are indeed able to discover other devices in the vicinity, but not as accurately as was hoped. In its attempts to break device discovery method's reliance on power, the system became reliant on

infrastructure, and other features that are not self-contained within the device. In particular, the system relies on Bluetooth devices to confirm that a device is within networking range. Reasons for the inaccuracy include a naive method of calculating the inter-entity distances. The system simply sums the intersection of two devices' sensed features to decide their inter-entity distance. This method was chosen based on the incorrect assumption that the devices would consistently sense the features of its location. The evaluations showed that three devices in the same location could each sense five Wi-Fi access points, and only share one of them. Other methods of calculating inter-entity distance do exist, and are worth investigating. Nonetheless, the large differences in the sensed features of one scene by three devices suggest that perhaps, relative location, and dynamic scene analysis are simply not reliable enough to perform device discovery in this manner.

The evaluations of the system's power use did not return results that were any more promising. Difficulties in battery use monitoring prevented any conclusive evidence from being collected, and it was shown that there may be no significant difference between performing a scan, and communicating with a server. If this result is accurate, then reducing the amount of time that a system spends while scanning will have no effect on the power consumption of the system.

The results of these two tests would indicate that the answer to the first research question is probably 'No.' but this answer is not conclusive.

6.1.2 Question 2

The inclusion of a server in the system meant that data transfer would need to occur. Cellular network data transfer was identified as being potentially expensive. Thus, the benefits gained in saving power would need to outweigh the costs of communicating with the server.

Evaluation revealed that, in general, the cost of communicating with the server was very low. On average, the amount of data transferred (1.8 kilobytes) was within two of the smallest billing units of a major cellular network service provider. And while the number of features in any given scan was positively correlated with the amount of data transferred, the strength of the correlation was weak. Thus, the answer to the second research question is 'yes'. The system can perform without using large amounts of data.

6.1.3 Other findings

The battery percentage tests, the CPU usage tests and the data transfer tests did reveal that other processes are operating on these devices all the time, in the background. Some of them use power, others use data. In some cases, the background use of power and data were so large it can be inferred that the consumption of resources in process of the GROUT device discovery system might be inconsequential to those used in the general running of the device. These findings call into question the relevance of trying to save power in the discovery process. When developing for complicated devices such as smartphones, perhaps one needs to address the problems that face smartphones as a whole, rather than merely those problems specifically related to the application in development.

6.4 Future work

As already stated, the method for calculating inter-entity distances is a naïve one. The literature review suggested other methods that utilized signal strength as well as the intersection in distance calculation (Frumm & Hinckley, 2004). Considering that not all features were sensed by a device in every scan, ways of obtaining more information from individual features could be researched.

The system is far from perfect. The results of the evaluations have shown that. But there exist other issues with the system that have not yet been addressed in this research. One of which is that there is no guarantee that a device that moves out of its last registered location will perform a scan so as to update its current context. This is a type of accuracy that was not considered at all during the evaluation. This problem certainly warrants further work, as any benefit that might be gained from the current system, could be lost if devices waste battery power attempting to connect to the veritable ghosts of devices; devices that are no longer where the servers claims they are. The incorporation of systems such as Sensless (Abdesslem, et al., 2009) could prove to be a useful way of having devices predict their own relocation, and preventing false positive discoveries.

6.5 Finally

While the results were not particularly fantastic, the research conducted here was not completely without merit. The system, as it exists, does have the potential to be a suitable replacement for some current device discovery systems. With some of the

suggestions in section 6.4, the accuracy of the distance measurements can be improved. This is merely one of many improvements that can be made to this system. All that remains to be done is to try.

References

Abdesslem, F.B., Phillips, A. & Henderson. (2009, August) Less is more: energy-efficient mobile sensing with senseless. *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds* (61-62). Barcelona, Spain.

Ableson, F. (2010, August 24). Using XML and JSON with Android, Part 1: Explore the benefits of JSON and XML in Android applications [Online article]. Retrieved from http://www.ibm.com/developerworks/web/library/x-andbene1/index.html on 13 October 2012.

Android API Reference. (2012, October 12). Settings.Secure API [Developer Documentation]. Retrieved from http://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID on October 15 2012.

Android Developer Guides. (2012a, October 12) Connectivity [Developer documantation]. Retrieved from http://developer.android.com/guide/topics/connectivity/index.html on 15 October 2012.

Android Developer Guides. (2012b, October 12) Platform versions [Developer documantation]. Retrieved from http://developer.android.com/about/dashboards/index.html#Platform on 20 October 2012

Agarwal, Y., Chandra, R., Wolman, A., Bahl, P., Chin, K. & Gupta, R. (2007, June) Wireless Wakeups Revisited : Energy Management for VoIP over Wi-Fi Smartphones. *Proceedings of the 5th international conference on Mobile systems, applications, and services* (179-191). PR, USA.

3Ananthanarayanan, G. & Stoica, I. (2009, June) Blue-Fi: enhancing Wi-Fi performance using bluetooth signals. *Proceedings of the 7th international conference on Mobile systems, applications, and services* (249-262). New York, NY, USA.

Anastasi, G., Conti, M., Gregori, E. & Passarella, A. (2008). 802.11 Power-Saving Mode for Mobile Computing in Wi-Fi hotspots : Limitations , Enhancements and Open Issues. *Wireless Networks, 14 (6)*, 745-768.

Bahl, P. & Padmanabhan, V. N. (2000, March) RADAR: an in-building RF-based user location and tracking system. *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*. (775-784). Tel Aviv, Israel.

Balasubramanian, N., Balasubramanian, A. & Venkatarami, A. (2009, November) Energy consumption in mobile phones: a measurement study and implications for network applications. *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (280-293). Chicago, IL, USA.

calum (2011, November 14) Answer to Can we programmatically know wifi status in windows phone 7? [Q&A forum]. Retrieved from http://stackoverflow.com/questions/8120100/can-we-programmatically-know-wifi-status-in-windows-phone-7 on 25 July 2012.

Cook, B., Buckberry, G., Scowcroft, I., Mitchell, J. & Allen, T (2009). Location by Scene Analysis of Wi-Fi Characteristics. Presented at the London Communications Symposium, University College London, England. Retrieved from http://www.ee.ucl.ac.uk/lcs/previous/LCS2006/2.pdf on 29 September 2012.

Crossen, A. & Budzik, J. (2006). Promoting Social Interaction in Public Spaces: The Flytrap Active Environment. In K, O'Hara & B, Brown (Eds.), *Consuming Music Together* (pp. 111 – 128). Dordecht, The Netherlands: Springer.

Cull, D. (2009) Key considerations in telecommunications regulation: An overview of the South African position [Online report].Retrieved from http://www.ellipsis.co.za/wp-content/uploads/2010/01/Key-Considerations-in-Electronic-Communications-Regulation-122009.pdf on 18 October 2012

FabioCreativity (2012, April 5) Neighboring Cell Info empty in XOOM 2 [Online forum post]. Retrieved from http://community.developer.motorola.com/t5/Android-App-Development-for/Neighboring-Cell-Info-Coming-empty-in-XOOM-2/td-p/25417 on 28 September 2012.

Farago, P. (2012, April 27) Social Networking Ends Games 40 Month Mobile Reign. [Blog post]. Retrieved from http://blog.flurry.com/bid/84512/Social-Networking-Ends-Games-40-Month-Mobile-Reign. on 12 May 2012.

Fitchard, K. (2012, October) The average US subscriber owns 1.57 mobile devices [Online article]. Retrieved from http://gigaom.com/mobile/the-average-us-subscriber-owns-1-57-mobile-devices/ on 24 October 2012.

Greenburg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R. & Wang, M. (2011, January/February). Proxemic interactions: the new ubicomp? *Interactions, 18,* 42-50.

Grepcode (2012) [Repository] Retrieved from http://grepcode.com/project/repository.grepcode.com/java/ext/com.google.android/android/ on 3 October 2012

Hall, E. S., Vawdrey, D. & Knutson, C. D. (2002) RF Rendez-Blue: Reducing Power and Inquiry Costs in Blutooth-Enabled Mobile Systems. *Proceedings of the 11th International Conference on Computer Communications and Networks*. (640-645).

Hall, E. T., Birdwhistell, R. L., Bock, B., Bohannan, P., Diebold, Jr. A. R., Durbin, M., ... , Vayda, A. P. (1968). Proxemics [and Comments and Replies]. *Current Anthropology, 9,* 83-108.

Havlicek, F. (2011, April 7). Re: Re; Neighboroughing *[sic]* cells never there. [Google groups discusion]. Retrieved from http://comments.gmane.org/gmane.comp.handhelds.android.devel/152313 on 28 September 2012.

Heathcliff74 (2012, March 12) Guides. [Developer documentation website]. Retrieved from http://www.wp7roottools.com/index.php/guides on 25 July 2012.

Hightower, J. & Borriello, G. (2001) A Survey and Taxonomy of Location Systems for Ubiquitous Computing. *Computer, 34 (8),* 57-66.

Hodkinson, P. & Lincoln, Sian. (2008). Online journals as virtual bedrooms? Young people, identity and personal space. *Young, 18*. 27-46.

HP (2002) Wi-Fi and Bluetooth – Interference issues. [Online research report]. Retrieved from http://www.hp.com/rnd/library/pdf/WiFi_Bluetooth_coexistance.pdf on 18 October 2012.

IDC (2012 August 8) Android and iOS Surge to New Smartphone OS Record in Second Quarter, According to IDC [Online press release]. Retrieved from http://www.idc.com/getdoc.jsp?containerId=prUS23638712 on 19 August 2012.

Inazaruk (2011, January 18) Using internal (com.android.internal) and hidden (@hide) APIs [Part 1, Introduction] [Online tutorial] Retrieved from https://devmaze.wordpress.com/2011/01/18/using-com-android-internal-part-1-introduction/ on 2 October 2012

Index mundi (Date unknown) Country Comparison > Telephones – mobile cellular per capita. [Online statistics website]. Retrieved from http://www.indexmundi.com/g/r.aspx?v=4010 on 25 October 2012.

Intel (2011, September 11) Intel® Power Profiler for Android* – A Power and Performance Related Data Profiling Tool for Android* Software Developers [Software Webpage]. Retrieved from http://software.intel.com/en-us/articles/intel-power-profiler-for-android-a-power-and-performance-related-data-profiling-tool-for on 1 October 2012.

Jes (2012, October 2). Cheapest ever mobile data prices in South Africa[Online forum discussion]. Retrieved from http://mybroadband.co.za/vb/showthread.php/470208-Cheapest-ever-mobile-data-prices-in-South-Africa on 15 October 2012

Json.org (date unkown) Introducing Jason [Online website]. Retreived from http://www.json.org/ on 20 October 2012.

Kerts (2010, June 26) [android-developers] neighboringsellinfo always null. [Blog]. Retrieved from http://androidgroup.blogspot.com/2010/06/android-developers-neighboringcellinfo.html on 28 September 2012.

Kinney, P. (2003, October) ZigBee Technology: Wireless Control that Simply Works. *Communication Design Conference*.

Koenig. R. (2010, November 27). Retrive every Neighbor CID on Android Platform [Online forum post]. Retrieved from http://www.anddev.org/retrieve_every_neighbor_cid_on_android_platform-t7449.html on 28 September 2012.

Krumm, J. & Hinckley, K. (2004). The NearMe Wireless Proximity Server. *Proceedings of the 6th International Conference on Ubiquitous Computing*. (283-300) Nottingham, England.

Lacey, M. (2012, March 26) Answer to How to get Wi-Fi and mobile network parameters in Windows Phone 7? [Q&A Forum]. Retrieved from http://stackoverflow.com/questions/9861789/how-to-get-wi-fi-and-mobile-network-parameters-in-windows-phone-7 on 25 July 2012.

Manweiler, J. & Choudry, R. R. (2011, June) Avoiding the rush hours: Wifi energy management via traffic isolation. *Proceedings of the 9th international conference on Mobile systems, applications, and services* (253-266). Bethesda, MD, USA, 2011).

Mentis, H. M., O'Hara, K., Sellen, A. & Trivedi, R. (2012) Interaction proxemics and image use in neurosurgery. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (927-936). Austin, TX, USA.

Microsoft Windows Phone 7 Development Centre (Date unknown) Application Platform Overview for Windows Phone. [API Documentation]. Retrieved from http://msdn.microsoft.com/library/windowsphone/develop/ff402531%28v=vs.92%29.aspx on 27 July 2012.

MSDN (2012, July 26) Application Policies. [Developer documantation]. Retrieved from http://msdn.microsoft.com/en-us/library/hh184841%28v=vs.92%29.aspx on 25 July 2012.

MTN (Date unkown) Save with MTN Internet bundles [Customer information website]. Retrieved from http://www.mtn.co.za/Internet/GetIt/Pages/InternetBundles.aspx on October 15.

Muller, R. (2012, April 4). Free Wi-Fi in Stellenbosch: High speeds and happy users [Online news article]. Retrieved from

http://mybroadband.co.za/news/wireless/47174-free-wi-fi-in-stellenbosch-high-speeds-and-happy-users.html on 17 October 2012.

Pering, T., Agarwal, Y., Gupta, R. & Want, R. (2006, June) Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces. *Proceedings of the 4th international conference on Mobile systems, applications and services* (220-232). Uppsula, Sweden.

Pering, T., Raghunathan, V. & Want, R. (2005, January) Exploiting radio hierarchies for power-efficient wireless device discovery and connection setup. *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design* (774-779).

PowerTutor (2009, November 17) PowerTutor: APower Monitor for Android-Based Mobile Platforms [Product Webpage]. Retrieved from http://ziyang.eecs.umich.edu/projects/powertutor/ on 1 October 2012.

Rahmati, A. & Zhong, L. (2007, June). Context-for-wireless: context-sensitive energy-efficient wireless data transfer, in *Proceedings of the 5th international conference on Mobile systems, applications and services* (165-178.) PR, USA.

Ramli, S.S.M. & Hasbullah, H. (2010, June). An improved inquiry procedure in device discovery Bluetooth network. *Proceedings of the 4th International Symposium on Information Technology 2010* (771-776). Kuala Lampur, Malaysia.

Reed, L., (2011, December 29) How can we improve the WPDev application platform? [Forum] Retrieved from http://wpdev.uservoice.com/forums/110705-app-platform/suggestions/1716881-bluetooth-data-transfer-apis on 27 July 2012

Rice, A. & Hay, S. (2010). Measuring mobile phone energy consumption for 802.11 wireless networking. *Pervasive and Mobile Computing, 6,* (6). 593-606.

Rubino, D. (2011, December 4). Native-code access "on the radar" for Windows Phone developers [News article] Retrieved from http://www.wpcentral.com/native-code-access-radar-windows-phone-developers on 25 July 2012.

Rubino, D. (2012, February 23) DFT releases Bluetooth file transfer utility for fully-unlocked phones [Homebrew]. [News article]. Retrieved from

http://www.wpcentral.com/dft-releases-bluetooth-file-transfer-utility-fully-unlocked-phones-homebrew on 25 July 2012.

Sorber, J., Banerjee, N., Corner M. D. & Rollins. S. (2005, June). Turducken: hierarchical power management for mobile devices. *Proceedings of the 3rd international conference on Mobile systems, applications, and services.* (261-274). Seattle, WA, USA

Spike66 (2011, March 24). Neighboroughing *[sic]* cells never there. [Google groups discusion]. Retrieved from http://comments.gmane.org/gmane.comp.handhelds.android.devel/152313 on 28 September 2012.

Sundholm, H. (2007). *Spaces within spaces: The Construction of a Collaborative Reality*. (Doctoral thesis). Available from Stockholm University Publications website. (Permanent link: http://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-6860)

Sundstrom, R. R. (2003). Race and place: Social Space in the production of human kinds. *Philosphy & Geography, 6.* 83-95.

Telecommunications act (1996). Telecommunications act, 1996 [No. 103 of 1996] – G 17581 [Online copy of legal document]. Retrieved from http://www.saflii.org/za/legis/num_act/ta1996214/ on 13 October 2012.

Tofel, K. C. (2010, Augst 18) Free Wi-Fi Hotspots Outnumber Paid Wi-Fi in the U.S. Says JiWire [Online news article]. Retrieved from http://gigaom.com/2010/08/18/free-wi-fi-hotspots-outnumber-paid-wi-fi-in-the-u-s-says-jiwire/ on 17 October 2012.

Want, R. & Pering, T. (2005, May). System challenges for ubiquitous & pervasive computing. *Proceedings of the 27th international conference on Software engineering* (9-14). St. Louis, MO, USA.

Yu, R. (2012, March 17) Verizon Wireless to end unlimited data plan. [Online news article] Retrieved from http://usatoday30.usatoday.com/tech/news/story/2012-05-16/verizon-wireless-unlimited-data/55028254/1 on 22 May 2012.

Zomm (2010). Zomm [Product website]. Retreived from www.zomm.com on 3 October 2012.

Appendix

A. Example JSON file output from Client

{"wifi":["00:26:5a:da:bd:1f"],"wifiID":"50:01:BB:3D:AC:C8","name":"Jill","cell":20511,"data":1907,"uID":"3e473e62d014f76e","wifiSize":1,"btSize":0,"btID":"50:01:BB:3D:AC:C7"}

B. Example File output from *Top* command

```
User 11%, System 11%, IOW 0%, IRQ 0%
User 12 + Nice 0 + Sys 12 + Idle 85 + IOW 0 + IRQ 0 + SIRQ 0 = 109

  PID CPU% S  #THR     VSS     RSS PCY UID      Name
12373   9% R     1    868K    412K  fg shell    top
  172   2% S    65 430428K  54664K  fg system   system_server
   60   1% S     1       0K      0K  fg root     synaptics_wq
   95   1% S    11   46020K   5948K  fg media    /system/bin/mediaserver
11490   1% S     7 276480K  20544K  fg app_74   skurro.firstsql


User 10%, System 6%, IOW 0%, IRQ 0%
User 12 + Nice 0 + Sys 7 + Idle 92 + IOW 0 + IRQ 0 + SIRQ 0 = 111

  PID CPU% S  #THR     VSS     RSS PCY UID      Name
12373   9% R     1    872K    424K  fg shell    top
   95   5% S    11   46020K   5948K  fg media    /system/bin/mediaserver
  172   1% S    65 430428K  54664K  fg system   system_server
    4   0% S     1       0K      0K unk root     watchdog/0
    5   0% S     1       0K      0K  fg root     events/0
```

C. Sample Output from Intel® Power Profiler for Android

(Intel ,2011).

```
BP 0,00 V 4 I 0 C 0 CPU0 832 12,46% 9,86% 2,59% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 24,00% 19,00% 5,00% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 9,90% 7,92% 1,98% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 30,30% 24,24% 6,06% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 30,00% 24,00% 6,00% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 13,86% 10,89% 2,97% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 15,15% 12,12% 3,03% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 2,97% 1,98% 0,99% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 1,01% 1,01% 0,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 4,95% 1,98% 2,97% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 16,83% 9,90% 6,93% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 2,02% 2,02% 0,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 1,98% 1,98% 0,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 2,02% 2,02% 0,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 4,00% 2,00% 2,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 8,00% 3,00% 5,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 8,00% 6,00% 2,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 2,97% 1,98% 0,99% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 1,01% 1,01% 0,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 4,00% 2,00% 2,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 8,91% 6,93% 1,98% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 3,00% 3,00% 0,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 1,02% 1,02% 0,00% CpuPStateResidency C
BP 0,00 V 4 I 0 C 0 CPU0 832 16,83% 13,86% 2,97% CpuPStateResidency
BP 0,00 V 4 I 0 C 0 CPU0 832 56,00% 46,00% 10,00% CpuPStateResidenc
```