PROVIDING SECURITY FOR MOBILE DEVICES USING C-BAC MECHANISM

N.Karthick Rajan, B.Tech ., M.E.,
Email: karthishiau@rediffmail.com

Abstract— In Android OS, installing malicious or non-market applications may result in privacy breaches and sensitive data leakage. Also Android OS often have access to sensitive data and resources on user device. By Default, Android users do not have control over the application once the applications have been granted the requested privileges during installation. The threat arises when a device application acts maliciously and it may leaks the user's personal data without the user's consent. In many cases, however, whether an application may get a privilege depends on the specific user context and thus, we need a Context-Based Access Control mechanism by which privileges can be specified by the user. The efficiency of our access control mechanism and the accuracy of context detection device resources to spy on the user or leak the user's personal data without the user's consent. C-BAC (Context- Based Access Control) mechanism is modified in an Android Operating System. Using C-BAC, requested privileges can be dynamically granted or revoked to applications based on the specific context of user. C-BAC policies, differentiates between closely located sub-areas within the same location.

Keywords*: Context-based access control, smartphone devices, security and privacy, policies, mobile applications*

1 INTRODUCTION

As smartphones are becoming more powerful in terms of computational and communication capabilities, application developers are taking advantage of these capabilities in order to provide new or enhanced services to their applications. For example, on March 2013 Samsung unveiled its Galaxy S4 device with 8 CPU cores and 9 sensors that enrich the device with powerful resources [1]. However, the majority of these resources can collect sensitive data and may expose users to high security and privacy risks if applications use them inappropriately and without the user's knowledge [2]. The threat arises when a device application acts maliciously and uses device resources to spy on the user or leak the user's personal data without the user's consent [3]–[5]. Moreover, users carrying their smartphones in public and private places may unknowingly expose their private information and threaten their personal security as they are not aware of the existence of such malicious activities on their devices. To prevent such threats, users must be able to have a better control over their device capabilities by reducing certain application privileges while being in sensitive contexts e.g. confidential meetings. To achieve this, smartphone systems must provide device owners with configurable policies that enable users to control their device usage of system resources and application privileges according to context, mainly location and time. Since such a feature is still missing in popular smartphone systems, such as in Android systems, it is crucial to investigate approaches for providing such control to device users.

The need for configurable device policies based on context extends from high profile employees to regular smartphone users. For example, government employers, such as in national labs [6], restrict their employees from bringing any camera-enabled device to the workplace, including smartphones, even though employees might need to have their devices with them at all times as their devices may contain data and services they might need at any time. With context-based device policies, employees may be allowed to use smartphones as they can disable all applications from using the camera and any device resources and privileges that employers restrict while at work, while the user device can retain

all its original privileges outside the work area. Context-based policies are also a necessity for politicians and law enforcement agents who would need to disable camera, microphone, and location services from their devices during confidential meetings while retaining these resources back in non-confidential locations. With context-based policies, users can specify when and where their applications can access their device data and resources, which reduces the hackers' chances of stealing such data.

Previous work on security for mobile operating systems focuses on restricting applications from accessing sensitive data and resources, but mostly lacks efficient techniques for enforcing those restrictions according to fine-grained contexts that differentiate between closely located subareas [7]. Moreover, most of this work has focused on developing policy systems that do not restrict privileges per application and are only effective system-wide [8]. Also, existing policy systems do not cover all the possible ways in which applications can access user data and device resources. Finally, existing location-based policy systems are not accurate enough to differentiate between nearby locations without extra hardware or location devices [7], [9], [10]. In most cases, such systems assume the context as given without providing or evaluating context detection methods of mobile devices [7], [11].

The design of context based policy systems for smartphones is challenging as it should fulfill the following requirements:

1) Applications should not be able to fake the location or time of the device, as they should not be able to bypass the policy restrictions applied on the device in a specific context.
2) As users are assumed to be mobile, the policy restrictions should be applied automatically on the device as the device's location changes.
3) The accuracy of location needs to be higher than the location accuracy by GPS, as we need to apply different policies in different spots or nearby sub-areas located within the same GPS location.
4) The enforcement of context-based policies should not require the application developers to modify source code, or impose any additional requirement on their applications.
5) The applied policy should not cause significant delays in the device functionality that could negatively impact the system performance.

In this paper, we propose a context-based access control (CBAC) mechanism for Android systems that allows smartphone users to set configuration policies over their applications' usage of device resources and services at different contexts. Through the CBAC mechanism, users can, for example, set restricted privileges for device applications when using the device at work, and device applications may re-gain their original privileges when the device is used at home. This change in device privileges is automatically applied as soon as the user device matches a pre-defined context of a user-defined policy. The user can also specify a default set of policies to be applied when the user is located in a non-previously defined location.

Configured policy restrictions are defined according to the accessible device resources, services, and permissions that are granted to applications at installation time. Such policies define which services are offered by the device and limit the device and user information accessibility. Policy restrictions are linked to context and are configured by the device user. We define context according to location and time. Location is determined basically through visible Wi-Fi access points and their respective signal strength values that allows us to differentiate between nearby sub-areas within the same work space, in addition to GPS and cellular triangulation coordinates whenever available. We implement our CBAC policies on the Android operating system and include a tool that allows users to define physical places such as home or work using the captured Wi-Fi parameters. Users can even be more precise by differentiating between sub-areas within the same location, such as living rooms and

bedrooms at home or meeting rooms and offices at work. Once the user configures the device policies that define device and application privileges according to context, the policies will be automatically applied whenever the user is within a pre-defined physical location and time interval.

2 ARCHITECTURE DESIGN

In this section, we introduce the design of our architecture through describing the components of our access control framework with the corresponding role of its entities.

Our framework consists of an access control mechanism that deals with access, collection, storage, processing, and usage of context information and device policies. To handle all the aforementioned functions, our framework design consists of four main components as shown in Figure 1.

The Context Provider (CP) collects the physical location parameters (GPS, Cell IDs, Wi-Fi parameters) through the device sensors and stores them in its own database, linking each physical location to a user-defined logical location. It also verifies and updates those parameters whenever the device is re-located.

The Access Controller (AC) controls the authorizations of applications and prevents unauthorized usage of device resources or services. Even though the Android OS has its own permission control system that checks if an application has privileges to request resources or services, the AC complements this system with more control methods and specific fine-grained control permissions that better reflect the application capabilities and narrow down its accessibility to resources. The AC enhances the security of the device system since the existing Android system has some permissions that, once granted to applications, may give applications more accessibility than they need, which malicious code can take advantage of. For example, the permission READ PHONE STATE gives privileged applications a set of information such as the phone number, the IMEI/MEID identifier, subscriber identification, phone state (busy/available), SIM serial number, etc.

The Policy Manager (PM) represents the interface used to create policies, mainly assigning application restrictions to contexts. It mainly gives control to the user to configure which resources and services are accessible by applications at the given context provided by the CP. As an example, the user through the PM can create a policy to enable location services only when the user is at work during weekdays between 8 am and 5 pm.

The Policy Executor (PE) enforces device restrictions by comparing the device's context with the configured policies. Once an application requests access to a resource or service, the PE checks the user-configured restrictions set at the PM to either grant to deny access to the application request. The PE acts as a policy enforcement by sending the authorization information to the AC to handle application requests, and is also responsible to resolve policy conflicts and apply the most strict restrictions.

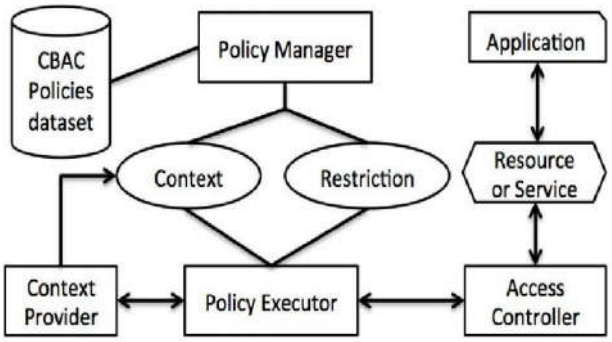

Fig. 2.1: Access Control Framework.

Through the PM, users can create CBAC policies through configuring application restrictions and linking them to contexts. When an application requests a resource or service, the AC verifies at run-time whether the application request is authorized and forwards the request to the PE. If the request is authorized, the PE then checks if there is any policy that corresponds to the application request. If such a policy exists, the PE requests from the CP to retrieve the context at the time of the application request. The PE then compares the retrieved context with the context defined in the policy. In case of a match, the PE enforces the corresponding policy restrictions by reporting back to the AC to apply those restrictions on the application request.

We carefully design the access control framework so that the user-configured policies are securely enforced with minimal processing steps and execution time to avoid any significant delays in responding back to the requesting application. As our design should securely handle policy execution, we maintain the context data provided by the CP to make sure it is accurate, precise and up-to-date.

3 IMPLEMENTATION

In this section, we introduce the technical details of our implementation which includes our modifications to the Android OS and the components of the Policy Manager custom application that acts as an intermediary between the OS and the user's desired policy configurations.

Policy Manager Components

The Policy Manager custom application consists of the four main Android application components: Activities, Broadcast Receivers, a Content Provider, and a Service.

Activities: The user interacts with the Policy Manager via activities, and through these activities, a user is able to define physical locations and subsequently configure a set of policies for these locations. The main constituents of these activities include Application Events, Permission Access, Resource Access, System Preferences, and Time Restriction.

Broadcast Receiver: We extended the Android's Broadcast Receiver class and created two custom classes, the Start Location Service Receiver and the Boot Receiver classes. The Start Location Service Receiver is responsible for triggering our customized Location Service for retrieving device location information. The Boot Receiver's main task is to schedule when the Start Location Service Receiver should request the location service. Once the Boot Receiver receives the BOOT COMPLETED Intent from the system, it uses the Android's Alarm Manager service to let the receiver schedule a pending Intent to be sent periodically to our Start Location Service Receiver in order to update the device location.

Service: The Location Service service is derived from the Intent Service class that facilitates offloading work from the main application's thread, allowing tasks to be performed in the background on a separate thread if desired. Location Service determines if the device has moved to or still is in a previously registered area. Offloading the aggregation of location-based data in a separate thread reduces the performance impact of the execution of the Location Service on the Policy Manager. We use the Alarm Manager to periodically activate the Location Service to ensure the device's location is always up-to-date. By default, the Location Service is activated once per minute, but we give the user the choice to configure how often the service is executed. The duration of the service depends on the number of snapshots of location parameters to be taken, which is currently configured to four per area.

Content Provider: The policies configured by the user are stored within the Policy Manager data directory. This data is private to our custom application and cannot be accessed by other applications or the system itself, as a result of Linux's kernel user ID access control mechanisms. PolicyCP is our custom content provider that acts as a secure intermediary between the policy database and all objects outside of the Policy Manager's running process. We chose to use the SQLite database to store user-configured policies due to the support and ease of programming provided by the Android API's associated with storing and managing databases on Android devices.

Permission Management

In the Android system, all resources that require explicit access rights in the form of permissions are protected by the Activity Manager Service class via permission verification.

When an application attempts to use any of these resources, the Activity Manager Service's method called check Component Permission is invoked to verify if the calling application has the appropriate permission(s) to access the resource.

We apply our modifications to this particular method by simply intercepting the permission call before the system performs its standard permission verification process. Given the permission and the application name, the system subsequently calls our custom content provider's revoke Resource Access to determine the next course of action. Depending on the user's policy configuration, the next course of action could either be returning the constant Package Manager. DENIED in the check Component Permission if the user has configured to block that permission from the requesting application, or letting the normal verification process take its course. We also give the ability to revoke any or all permissions system-wide via the Policy Manager's interface.

Restrictions on User Data

Our implementation of data obfuscation complements much of the techniques used in [16] and [17], but instead under the domain of CBAC policy restrictions. We obfuscate user data from applications attempting to access it if the policy restriction applies to those applications. We modify the Android APIs that access the user data saved on the device.

Relational database systems are the common data management systems used to create, store, and manage user data. Accessing these data usually require calling the Content Resolver's query() method, and thus we modify it for our purposes. Instead of returning the expected Cursor object needed to point to the required data, a Null Cursor object is substituted. A Null Cursor object represents an empty dataset, such as an empty list of pictures as if pictures were not present or never stored on the device.

Managing System Peripheral State

We also give users the option to configure a policy to restrict access to peripherals (e.g. Bluetooth) when entering a particular location. Specifically, users can set up their devices to prevent applications from modifying a peripheral's current state (enabled/disabled). While it is possible to modify a peripheral's current state by using permission management, we modify the specific methods that enable/disables these peripherals in order to prevent applications from crashing that do not have code for handling exceptions resulting from revocation of permissions. As an example, for Bluetooth we modified the Bluetooth Adapter class and for Wi-Fi we modified the Wifi Manager class so to assure that these modifications do not result in application crashes and to prevent applications from modifying peripherals current state. Whenever an application tries to modify the state of a system peripheral, our content provider Policy CP checks the validity of the request and would refuse the request if the request tries to override a user-configured restriction.

4 CONTEXT MANAGEMENT

The main source of location-related information for our access control system is the Wi-Fi APs and their corresponding signal strengths. Location information acquired from GPS and cellular towers is also aggregated to our context definition but may not be sufficient for indoor localization especially that they may become weak or unavailable inside buildings or areas within building structures [18], [19]. However, location information retrieved from Wi-Fi parameters could be more precise to differentiate between closely located sub-areas within the same GPS location [20], [21].

A spatial region is represented by combining GPS coordinates, cellular triangulation location data, and Wi-Fi APs and signal strengths. In Android, the GPS coordinates and cellular triangulation are obtained in a similar fashion by invoking the Android Location Manager service. Once the Location Manager is invoked, we request location updates by calling the request Location Updates method that returns a Location object which contains latitude, longitude, timestamp, and other information.

Wi-Fi is handled differently than the previous two location methods. We obtain the Wi-Fi parameters by invoking the Wifi Manager service to retrieve the Wi-Fi access points scans. We register our Broadcast Receiver Wifi receiver with an Intent Filer action to receive the broadcasted Wi-Fi scanned intent, and then request for and subsequently process the actual scanned access points data. In our CBAC policy system, we provide users with a utility to define physical locations by either capturing snapshots of location data of the desired areas or by manually entering the area location coordinates. In the following sections, we show our design and implementation of the location capturing phase when users define and store physical locations, and the location detection phase when device detects its location and match it with a pre-defined policy context.

Location Capturing Phase

Figure 2 describes how location data is captured for each context defined by the user. Through the location scan interface, the user is able to capture several snapshots of location data in different sub-areas. For each sub-area, location data is accumulated from each snapshot; the GPS coordinates and the cellular triangulation, when applicable, import the latitude and longitude from the captured snapshots and only select those with the highest position accuracy. With respect to Wi-Fi, we noticed that the Wi-Fi access points signal strengths fluctuate even if the device is stationary or motionless.

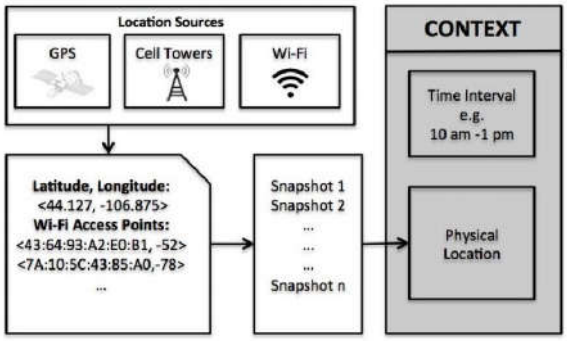

Fig. 4.1: Location Capturing Phase.

Therefore, our application scans the signal strengths of each access point for several seconds gathering the RSSI values at each particular sub-area. Finally, the accumulated data, which mainly consists of Wi-Fi access points with signal strength ranges in addition to GPS and cellular triangulation data as supporting location information, will represent one physical defined location to the user.

`Any location that is not defined by the user or does not have location information saved on the device will be considered "Unregistered". Therefore, we designate a default policy restrictions for the user to configure whenever the device is located in an unregistered location. In addition, we allow users to register locations that have not been previously visited. This is achieved through either manually entering the publicly known longitude and latitude of the desired location, or by acquiring the fine-grained Wi-Fi parameters from other devices who have saved those parameters. This becomes very practical when the user is switching between two devices and needs to import previously saved policy contexts to the new device.

Our implementation does not store all the GPS or triangulated cellular coordinates acquired, rather a subset of those coordinates that bound into a convex hull and their associated precision. The points in the interior of the convex hull are discarded. We also only store the RSSI range for each distinct Wi-Fi access point scanned. This range is the minimum and maximum RSSI values aggregated from all the sub-areas for each access point. A sub-area is therefore represented as a range of Wi-Fi signal strength values at the least, and if with high position accuracy, also a representation of a convex hull of GPS or triangulated cellular coordinates.

Location Detection Phase

Figure 3 describes how device context is detected and matched with pre-defined context. Periodically, the location background service is re-instantiated to accumulate location-context data to determine the device's current whereabouts. Like when registering and scanning a sub-area in the location capturing phase, we scan the device's location-related data. The list of user-registered areas that have a subset of the scanned neighboring access points are extracted from the database first. Matching distinct access

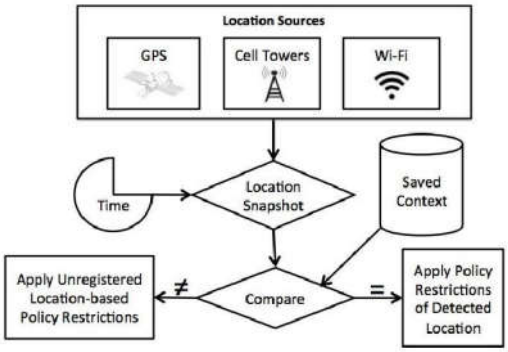

Fig. 4.2: Location Detection Phase.

Points is computationally less expensive than determining if coordinate position falls within the boundaries of a convex hull. Then, using the current signal strengths of the access points, we reduce the list to only a set of "best-match" list of physical locations whose access points fall within the current captured signal strength values.

5 EXPERIMENTAL RESULTS

In this section, we report experimental results about the CBAC mechanism and evaluate its impact on the device system and applications. Our modifications to the Android source code were tested on the Android Nexus 4 cellular device and Android Nexus 7 tablet running the Android

4.2.2 OS (API level v. 17). We ran the top 250 applications from the Google Play market for testing and evaluating our modifications. Each experiment has been carried out with the help of the Android Debug Bridge (ADB) utility by using the command "adb logcat".

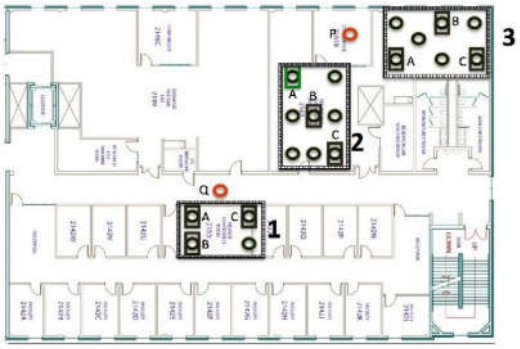

Fig. 5.1: Tested areas in one of our campus buildings.

Experiment 1: Location Detection Accuracy. The goal of this experiment is to evaluate the accuracy of the location detection algorithm used in our CBAC mechanism. We measure the number of success and failure detections per sub-area. Figure 5.1 displays the schematics of one building where we performed some of our experiments. The large, grid-pattern rectangles point out main locations or areas, identified by numbers. Areas outside the rectangles are considered "unregistered." Figure 5.1 shows three tested rooms located on the same floor. In each room, we chose at least four spots to participate in the location capturing phase to accumulate location-related data, in order to construct a robust set of location parameters per room to be stored in the database. In each location, we analyzed three sub-areas, indicated by 'A', 'B', or 'C' and measure the detection rate in each of these subareas.

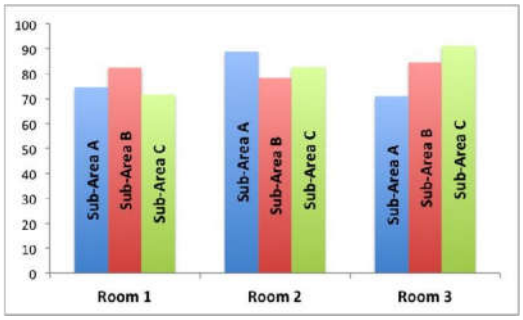

Fig. 5.2: Detection accuracy rate of closely located areas.

Figure 5 displays the detection accuracy rate in the 3 sub-areas of rooms 1, 2 and 3. At each of the sub-areas of each room, we performed 50 location tests and counted the number of successful detections. Our experimental results show that the successful detection rates were up to 91%, and in the worst case scenario we had up to 29% of incorrect detections. This experimental result was complemented by testing several "unregistered" areas around the registered rooms. We detected 16% of false positives, that is, unregistered areas that appeared to be user-defined. Within the registered areas, the values of the signal strengths of matching Wi-Fi access points fell within the range of signal strengths first acquired during the location capturing phase.

Experiment 2: RAM Performance Overhead. The purpose of this experiment is to evaluate the timing overhead

TABLE 3: Time overhead (in milliseconds) for some of the core Android methods that were modified.

Method	Overhead
checkComponentPermission(..)	12.220
Intent-startActivity(..)	12.708
Intent-startService(..)	5.402
Intent-sendBroadcast(..)	5.208
User Data-ContentResolver(..)	12.300
Device Peripherals-setEnable(..)	8.351

6 CONCLUSION AND FUTURE WORK

In this work, we proposed a modified version of the Android OS supporting context-based access control policies. These policies restrict applications from accessing specific data and/or resources based on the user context. The restrictions specified in a policy are automatically applied as soon as the user device matches the pre-defined context associated with the policy. Our experimental results show the effectiveness of these policies on the Android system and applications, and the accuracy in locating the device within a user-defined context.

Our approach requires users to configure their own set of policies; the difficulty of setting up these configurations require the same expertise needed to inspect application permissions listed at installation time. However we plan to extend our approach to give network administrators of organizations the same capabilities once a mobile device connects to their network.

REFERENCES

[1] Wikipedia, "Samsung galaxy s4 specifications," http://en.wikipedia. org/wiki/Samsung Galaxy S4, May 2013.

[2] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in Proceedings of the 9th USENIX conference.

[3] J. Leyden, "Your phone may not be spying on you now - but it soon will be," http://www.theregister.co.uk/2013/04/24/kaspersky mobile malware infosec/, April 2013.

[4] R. Templeman, Z. Rahman, D. J. Crandall, and A. Kapadia, "Placeraider: Virtual theft in physical spaces with smartphones," CoRR, vol. abs/1209.5982, 2012.

[5] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones," in Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS), Feb. 2011.

[6] L. L. N. Laboratory, "Controlled items that are prohibited on llnl proporty," https://www.llnl.gov/about/controlleditems.html.

[7] M. Conti, V. T. N. Nguyen, and B. Crispo, "Crepe: context-related policy enforcement for android," in Proceedings of the 13 th international conference on Information security, ser. ISC'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 331–345.

[8] A. Kushwaha and V. Kushwaha, "Location based services using android mobile operating system," International Journal of Advances in Engineering and Technology, vol. 1, no. 1, pp. 14–20, 2011.

[9] S. Kumar, M. A. Qadeer, and A. Gupta, "Location based services using android," in Proceedings of the 3rd IEEE international conference on Internet multimedia services architecture and applications, ser. IMSAA'09, 2009, pp. 335–339.

[10] M. S. Kirkpatrick and E. Bertino, "Enforcing spatial constraints for mobile rbac systems," in Proceedings of the 15th ACM symposium on Access control models and technologies, ser. SACMAT '10. New York,NY, USA: ACM, 2010, pp. 99–108.

[11] A. Gupta, M. Miettinen, N. Asokan, and M. Nagy, "Intuitive security policy configuration in mobile devices using context profiling," in IEEE International Conference on Social Computing, ser. SOCIALCOM-PASSAT '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 471–480.

[12] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," Security Privacy, IEEE, vol. 7, no. 1, pp. 50–57, 2009.

[13] E. Trevisani and A. Vitaletti, "Cell-id location technique, limits and benefits: an experimental study," in Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on, 2004 , pp. 51–60.

[14] J. LaMance, J. DeSalas, and J. Jarvinen, "agps: A low-infrastructure approach," http://www.gpsworld.com/ innovation-assisted-gps-a-low-infrastructure-approach/, March '02.

[15] "Sky hook," http://www.skyhookwireless.com/.