An Empirical Study of Location Truncation on Android

Kristopher Micinski, Philip Phelps, and Jeffrey S. Foster
Computer Science Department, University of Maryland, College Park, MD 20742, USA
{*micinski, pdphelp, jfoster*}*@cs.umd.edu*

***Abstract*—Location-based apps are popular on Android, but they raise a number of privacy concerns. In this paper, we empirically study how one popular location-privacy enhancing technique, location truncation, affects app utility. That is, we try to answer the question: How much can we truncate the location information given to an app while still preserving the app's usability? To this end, we developed CloakDroid, a tool that can apply a user-specified amount of location truncation to an arbitrary Android app. We then surveyed a large set of apps from Google Play and found that many apps use location to generate lists of nearby objects. For this category of apps, we developed three metrics of the usability of an app's output list before and after location truncation: Edit distance between the lists; set intersection size, treating the two lists as sets; and the additional distance incurred to go to the first (closest) entry on the list. Finally, we used CloakDroid to conduct experiments to compute these metrics for six apps across a range of locations (in cities varying in population from 6,000 to 8.2 million) and truncations (from 0.1 km to 50 km). We found that the amount to which an app's location could be truncated mainly depends on the density of objects being measured by the app, and that most apps can have their inputs truncated to between 5 km and 20 km without significantly degrading app utility.**

I. Introduction

Many mobile apps use location information. As implemented today, such apps typically send the device location over the Internet, and hence users may be understandably concerned about potential privacy violations. In response, researchers have developed a number of location privacy-enhancing mechanisms [1], [2], [3], [4], [5], [6], using a range of analytical models to estimate the degree of privacy and anonymity achieved.

However, there has been much less work [7] on understanding how location privacy-enhancing technology affects the *utility* of apps—that is, does an app still work well enough even when location privacy is increased. In this paper, we tackle this question by *directly* measuring how *location truncation*—quantizing the current location to a user-specified grid spacing—changes the output of a range of Android apps.

Location truncation is one of the more realistic approaches to location privacy, with several advantages: It is easy for users to understand and therefore trust. It can be implemented easily and efficiently. And it does not require foreknowledge of the set of locations to be visited. There are also some disadvantages: Location truncation primary defeats localization attacks, rather than deanonymization or tracking [8]. Location truncation may also be overcome through a combination of prior knowledge (e.g., likely movement speeds, locations of road networks, known habits, etc) and repeated queries over a long period of time [9] [10]. Even so, we believe that for many everyday uses, these limitations are not severe for typical users.

While location truncation potentially increases privacy, it may not be ideal for all apps (e.g., it defeats turn-by-turn navigation), and its effects may be hard to measure (e.g., for apps that use location to decide which ads to show). We retrieved the top 750 apps across all categories of the Google Play store, and found that 275 of these use location. We ran each app to determine how and why they use location, and found six main categories: ad localization, listing nearby objects, fine-grained tasks (e.g., turn-by-turn navigation), geotagging content (such as pictures), finding the nearest city, and getting local weather information. (See Section II.)

Based on this result, we chose to study the effect of location truncation on apps that produce lists of nearby objects. These apps are a good target for a variety of reasons: First, the effect of truncation is easy to determine, as we can simply look at the output list. Second, truncation is plausibly useful for these apps: on Android, most apps do not include internal maps, but rather use Android's Intent mechanism to ask the Google Maps app to do any necessary mapping (this was the case for all of the apps in our study). Thus, truncating location will help increase privacy of the user to the subject app, but will not affect the ultimate usage of the information, assuming that Google and Google Maps are trusted. Finally, location truncation gives users a very clear tradeoff, in which they can go a little out of their way in exchange for more privacy.

We implemented location truncation in CloakDroid, a tool built on top of the Dr. Android and Mr. Hide framework of Jeon et al. [11]. CloakDroid modifies a subject app to use a special location service that snaps reported locations to a latitude/longitude grid whose spacing is specified by the user. (See Section III for details of CloakDroid.)

To evaluate how CloakDroid affects app utility, we applied it to six Android apps and measured how their output changed under a range of conditions. We developed three different metrics to judge utility of output: The *edit distance* of the nominal list under truncation compared to the reference list without truncation; the *additional distance*, which computes how much farther away the first list item in the

nominal list is compared to the reference list; and the *set intersection size*, which counts the elements common to the reference list and nominal list.

We ran each app against 10 randomly chosen locations spread across 6 regions of various sizes, ranging from New York, NY, population 8.2 million, to Decatur, TX, population 6,000. For each location–region pair, we varied the truncation grid spacing from 0.1 km to 50 km and measured the result. We found that the under the additional distance metric, there is typically no change in app utility up to 1 or 2 km truncation, and that if the user is willing to go up to 1 km out of their way, most apps can sustain truncation amounts of 5 km or more. Additionally, the most important determinant of acceptable truncation was the density of the objects listed by the app—the lower the density, the more location could be truncated before seeing an effect. Across all metrics, most of the subject apps were able to have their inputs truncated to between 5 km and 20 km without significantly degrading app utility.

To our knowledge, we are the first to directly and systematically study how location truncation affects app utility, and our results suggest location can be truncated to a significant degree without compromising app utility.

II. How Apps Use Location

We began our study by trying to understand how apps use location in practice. We downloaded the top 750 most popular free apps across all categories of the Google Play store as of April 2012, and found that 275 of these apps request location permission. We installed each app on a device or emulator and used the app, navigating through its screens until we could categorize how the app uses location information. We identified the following categories of location usage, summarized in Figure 1:

- Ads — Most ad networks deliver location-targeted ads to users, and ads are very common on Google Play apps. Thus, this was the most popular category of location usage. However, this category is not a good subject for our study, as it is unclear how to measure the effect of location truncation on ad selection. In Figure 1, the first row counts apps that use location only for ads. Often apps use location both for ads and another purpose; the count of these apps is called out separately in the remaining rows.
- Listing nearby objects — The second most common category of location usage is to find some set of objects that are near the user's current location. For example, an app might inform the user of nearby traffic accidents or construction to avoid. This is the category we selected for our study.
- Fine-grained targeted content — Many apps require very fine-grained location information to provide their service, e.g., an app that remembers where the user parked. Location truncation is not likely a good idea for these apps.
- Content tagging or check-in — Location information is often used for geotagging data (e.g., pictures or social network posts) or "checking in" at the current location. For these apps, location truncation is plausible, but it is difficult to measure the effect on utility.
- Nearest city — Several apps only need to know the current location to a very coarse degree, e.g., Living Social and Craigslist just need to know the nearest city to select the right data to display to the user. Truncating location for these apps will have essentially no effect, as they already use coarse data (though clearly it might be good from a security perspective).
- Weather — Weather apps use the current location to choose what weather data to report. We briefly explored the effect of truncation on these apps, but found two key problems: First, by nature, small location changes usually have no effect on weather; and second, weather data changes over time, and so it is hard to get consistent data sets because our experiments can take a few hours to run for each app.
- Unsure — There were 17 apps for which we could not determine why or how they use location information. These may be cases of over-permission, app functionality that we missed in our testing, or possibly malicious information gathering.
- Unable to test — Finally, we could not test 52 apps because they required either telephony-specific features not available on our test devices (which did not have phone or data plans) or subscriptions to paid services. For example, many vendors offer apps that work only on phones connected to their network.

	w/ ads	Total
Only ads		58
Listing nearby objects	6	43
Fine-grained uses	7	34
Content tagging	4	28
Nearest city	1	22
Weather	2	21
Unsure		17
Unable to test		52

Figure 1: Location information usage in the top apps

Based on the results of this study, we decided to investigate the effect of location truncation on apps that list nearby objects, as this category is common and it is clearly meaningful to measure the result of truncation on these apps.

III. Implementation

An Android app consists of its bytecode, written in Java but compiled to Google's Davlik bytecode format [12], along with configuration information and other app resources. The various components of an app are combined together into

an application package file, or apk, which is then digitally signed, after which it can be installed on a mobile device.

Apps that use location information must request ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permission at install time; the former allows access to a cell-tower–based location and the latter to a GPS location (if available). FINE_LOCATION access is provided at full device resolution, and COARSE_LOCATION access is truncated to 200 m or 2 km resolution, depending on the device parameters.[1] All the apps in our experiments use FINE_LOCATION.

As mentioned in the introduction, we implemented location truncation in the form of CloakDroid, a tool that changes a subject app to receive modified location information. Programmatically, location access on Android goes through a fairly narrow API, which makes it easy for CloakDroid to intercept. Apps first request an instance of class LocationManager, which includes, among others, methods to retrieve the last known location and to register a handler for location updates. CloakDroid controls the granularity of location information by modifying the subject app so that, instead of using Android's LocationManager, it uses a replacement class provided by CloakDroid. That class delegates all calls to the system's LocationManager, but adds a user-specified amount of truncation before returning coordinates.

We implemented CloakDroid using Dr. Android and Mr. Hide [11], a prior system that can replace permissions like ACCESS_*_LOCATION with finer-grained permissions. In the original Dr. Android and Mr. Hide paper, we proposed a permission that truncates to a fixed amount (150 m); in this paper, we experiment with a much wider range of truncations and evaluate the results much more thoroughly.

Dr. Android and Mr. Hide work by transforming the (Dalvik) bytecode of an app. First, the app's apk file is unpacked, which yields, among other files, the classes.dex bytecode for the app. That bytecode is then parsed, and references to the location manager are modified to refer to our replacement location manager, whose code is also concatenated to the bytecode file. The app is then repacked into a new apk and signed with a new key.[2]

The standard version of Dr. Android and Mr. Hide also removes the original location permissions from the app and replaces them with permission to talk to the Mr. Hide service at run time, which carries out location queries on the app's behalf. In our implementation, we keep the Mr. Hide service, but leave the original location permissions in place, as several of our subject programs have code that checks for those permissions and aborts if they are removed. (We could eliminate those permission checks, but for purposes of our study there is no reason to.)

[1] https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/location/java/android/location/Location.java

[2] Only signed apps may be installed on a phone. However, the signature only serves to indicate trust relationships between apps, and is not important for our subject apps.

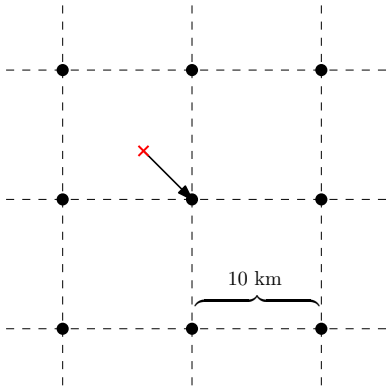


Figure 2: The user (represented by the red cross) has their location truncated to a grid of fixed points.

Once we can intercept calls that pass location information to the app, it is easy to modify the coordinates however we wish. For our experiments, we truncate location information, snapping to a user-specified grid, as illustrated in Figure 2. Formally, our implementation truncates to a grid with spacing s in kilometers using the formulae:

$$lat' = s_{lat}\lceil lat/s_{lat}\rceil \quad long' = s_{long}\lceil long/s_{long}\rceil$$

where *lat* and *long* are the actual latitude and longitude, and their primed versions are the new coordinates. Here s_{lat} and s_{long} are the grid spacing translated into degrees of latitude and longitude, respectively. For latitude we use the WGS 84 approximation for North America [13]:

$$s_{lat} = \frac{s \text{ km}}{111.5 \text{ km/deg}}$$

For longitude we use a standard approximation [14]:

$$s_{long} = s \text{ km} \cdot \frac{180\sqrt{(1 - e^2 \sin^2(\phi))}}{\pi a \cos(\phi)}$$

where $e^2 = (a^2 - b^2)/b^2$ is the eccentricity of Earth, ϕ is the latitude, a is the radius to the equator, and b is the radius to the poles.

We verified that CloakDroid works correctly in two ways. First, we inserted logging code in CloakDroid to give the original position and position after location truncation. We then took the set of locations from our testcases, truncated them to varying amounts, and then verified the resulting positions were correct. Second, we confirmed that our subject apps' behaviors changed in sensible ways as we varied the location truncation amount.

The Android location manager API also can provide speed information, which our implementation truncates speed using a similar formula. However, none of our subject apps use speed information. Moreover, device-reported speed is often unreliable, so many apps ignore it, preferring instead to estimate speed using successive location fixes.

Name	Objects in List
Gasbuddy	Gas stations
Restaurant Finder	Restaurants
Hospitals Near Me	Hospitals
WebMD	Pharmacies and clinics
Walmart	Stores
TD Bank	ATMs and branches

(a) Descriptions of each subject app.

City	Population	Radius (km)
New York, NY	8,200,000	30
Dallas, TX	1,200,000	20
New Haven, CT	130,000	10
Baltimore, MD	620,000	12
Redmond, WA	54,000	4
Decatur, TX	6,000	4

(b) Selected population centers.

0 km	0.1 km	0.2 km	0.5 km	1 km
2 km	5 km	10 km	20 km	50 km

(c) Truncation amounts tested.

Figure 3: Experimental parameters.

IV. Experimental Design

We conducted a systematic study of the effect of location truncation on apps that list nearby objects, the largest category in Figure 1 for which location truncation's effect is clearly measurable. We selected a variety of such apps, listed in Figure 3a, from Google Play. We chose apps that rely on a variety of different data sets and run under our tool chain with minimal difficulty. We then modified each app using CloakDroid, as outlined in Section III. Next, we randomly chose a set of locations and then captured the output list of each app as we varied the current location and the amount of truncation applied to it. Finally, we used several metrics to estimate how each truncation affected the app's utility.

A. Locations and truncations

In a pilot study, in which we examined the results of CloakDroid on a variety of apps, locations, and truncations in an ad hoc manner, we noticed that truncation has quite different effects depending on where apps are run. For example, there is a significantly higher density of restaurants in a big city than in a rural area, causing one of our subject apps, Restaurant Finder, to behave very differently under truncation in each locale. Thus, in selecting locations for our study, we chose them from a range of population centers of varying sizes, as shown in Figure 3b.

We modeled each city as a circle centered on the geographical city center with a radius estimated from looking at a map of the region. In our pilot study, we also found that many apps produce error screens if given a location that cannot be mapped to a real address, e.g., if the current

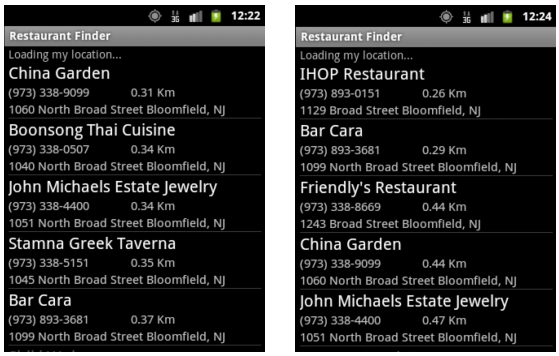


Figure 4: Restaurant Finder without (left) and with (right) 0.2 km of truncation.

location is in the ocean or a heavily forested area. Thus, we discarded any random points that caused problematic outputs from our apps, and randomly picked new points until we had a set of 10 points that produced correct output across all apps when the apps were run with no truncation.

For each random location, we ran each app with 10 different truncation amounts (including no truncation), listed in Figure 3c. We chose an upper limit of 50 km because many of our subject apps give results up to or less than 50 km, so the metrics would be undefined beyond that level.

B. Metrics used for evaluation

Recall from Section II that we identified apps that list nearby objects as the best candidate for our study of location truncation. For example, Figure 4 gives two screenshots of the Restaurant Finder app being run on a location in New York, NY (close to Bloomfield, NJ) both without and with truncation. Here the lists we consider are the restaurants, in order from closest to farthest, along with their distances.

We explored three ways to measure location truncation's effect on the output list of such apps: edit distance of the changed list from the original, the size of the intersection of the original and changed lists, and the additional distance of the first (closest) list item on the changed list.

Edit distance: The first metric we investigated is *edit distance*, which is the number of edits (insertions, deletions, or swaps) needed to change one list into another [15]. For example, the edit distance between the lists in Figure 4 is five, because every item in the list has changed.

Like the other apps we studied, Restaurant Finder returns more than one screenful of objects. Thus, in performing our measurements, we need to decide how much of the app's output to use. We could pick only the first screen of data, but this restricts edit distance to just a few possible values. We could pick all the results, but that has the disadvantage that there could be many changes in the long tail of a list, yet few users would bother looking that far. Thus, as a compromise, we opted to compute edit distance over the first four screens of an app, which we think covers typical usage patterns

while providing useful data. In each of the apps we tested, this corresponded to the first twenty list elements.

Set intersection size: Edit distance was the first metric we thought of, but when we tried measuring it, we found it to be problematic: Especially on the first screen of data, many locations are very close, e.g., on the left of Figure 4, distances range from 0.31 km to 0.37 km. Yet edit distance would count reorderings of those locations in the metric.

Thus, we also explored using set intersection size as a metric. We ignore ordering and compute how many objects are in both the original and changed list. For example, in Figure 4, the set intersection has size three. As with edit distance, we compute the intersection on the list displayed across the first four screens of output.

The set intersection metric is also more natural than edit distance in that it corresponds directly to a common user task of checking whether some particular object is nearby. For example, we might use Restaurant Finder to ask, is there any McDonalds nearby, or we might use GasBuddy to look for the closest Shell station.

Additional distance incurred: Ideally, we would like to measure how actual users' behavior is affected by location truncation, but this is difficult to do without a large number of human participants. Instead, we developed an *additional distance* metric that computes how much farther a user who always picks the first list item would go given the changed list compared to the original list.

For example, in Figure 4, the closest restaurant is actually China Garden, 0.31 km away. However, under truncation, the closest restaurant appears to be an IHOP. While the IHOP's reported distance is 0.26 km, that is the result from the truncated location—we need to look at the original output list to find out how far away the restaurant actually is. In this case, IHOP appears on the second screen of the original list, and it is 0.42 km away. For this example, then, the additional distance is $0.42 - 0.31 = 0.11$ km.

Restating this more generally, let $D(X)$ be the distance to object X on the original list. Then the additional distance of a changed list is $\Delta = D(\textit{First original}) - D(\textit{First changed})$, where *First original* is the first item of the original list and *First changed* is the first item of the changed list. Notice that this metric may be undefined if *First changed* does not appear on the original list. To reduce the chances of this, we consider all output screens of an app in computing the metric, and not just the first four screens. However, in our experiments the metric is still sometimes undefined for some apps under large truncation amounts.

C. Testing infrastructure

To compute the results of our experiments, we needed to run each subject app at 60 locations and 10 different truncation amounts. This is clearly far too many configurations to run by hand. Instead, we developed testing infrastructure to let us run each configuration automatically, scrape the resulting output lists, and then run our metrics on the result.

The core technology we used is Troyd [16], an open source, black box testing framework that can execute apps (e.g., launch them, click buttons, enter text boxes, scroll the screen, etc.) and gather text from the GUI. Troyd allows us to write a high level test case (e.g., to click a sequence of buttons and gather some text) that can be reused for various test configurations. We developed a server that takes an app modified by CloakDroid; resigns it with a shared user ID so that it can be controlled by instrumentation; and then runs testcases against the app using Troyd.

We ran apps on both Google Nexus S devices and on device emulators; our infrastructure works the same in both cases, and performing some runs on actual devices acted as a sanity check. In all cases, the Android platform was Gingerbread 2.3.3 with the Google APIs, such as maps, installed. These APIs are special dynamic libraries licensed by Google and not included on plain Android distributions; these APIs are needed for the subject programs to run.

We found that, while Troyd mostly did what we wanted, we needed to extend it to handle additional GUI elements. For example, while Troyd can capture the currently visible GUI elements, many Views (such as those in ListView elements) are reused, and their contents only appear when the screen is scrolled. Thus, we extended Troyd with special functionality to navigate to screens that may have unrendered information. We also found that, as is usual in large-scale testing, experimental tests sometimes failed unpredictably, e.g., an emulator or device would crash for no apparent reason. These are likely due to subtle bugs lurking in the OS or emulator implementation that were tickled by the experiment's atypical usage pattern. Thus, our testing infrastructure catches any failed runs and reschedules those experiments to be run again. Finally, our testing infrastructure allows multiple experiments to be done in parallel; the majority of the experiments were done on a machine with six Intel Xeon processors, each with four cores, and 48 GB of RAM. Each individual test took on the order of a few minutes to run, depending on the app. Our tests ran up to fifteen emulators at once.

V. Results

Finally, we present the results of our study. At a high level, we found that across our subject apps and locations, location can be significantly truncated without significantly harming utility: up to 20 km when an app is used in a lower population area, and usually (for five of six apps) at least 5 km in more populous areas. We also found that apps have relatively little change in utility up to a certain amount of truncation, typically about 5 km, after which the app's output becomes much less usable. Finally, we found that the factor that most determines the ability to truncate location was the density of the set of locations the app computes over.

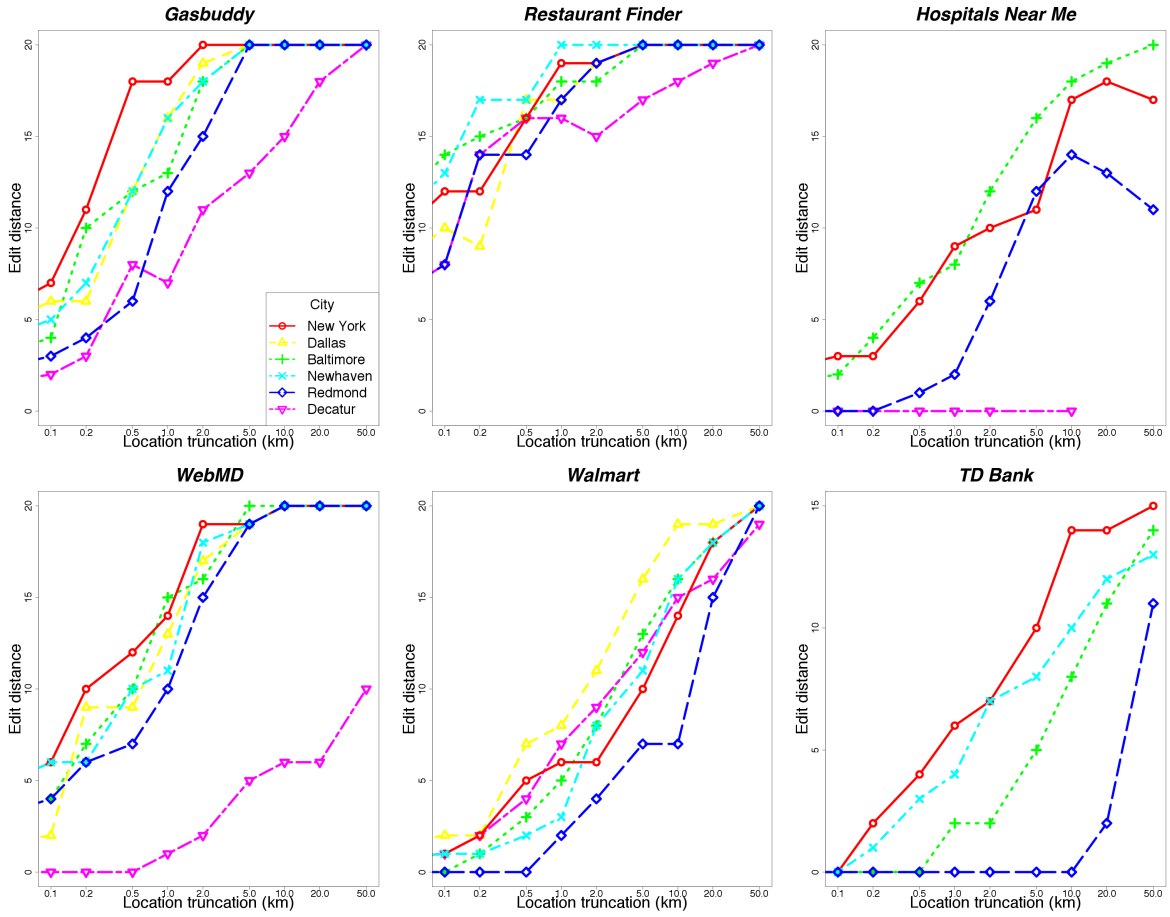


Figure 5: Graphs of median edit distance versus truncation amount. Higher edit distance implies lower utility.

In the discussion that follows, we include several plots. In each plot, the x-axis is the truncation amount; note that because of our choice of truncations, this axis is essentially log-scale. The y-axis is the median of one of our metrics on the 10 randomly chosen points for each population center.

We next consider each of our three metrics in turn.

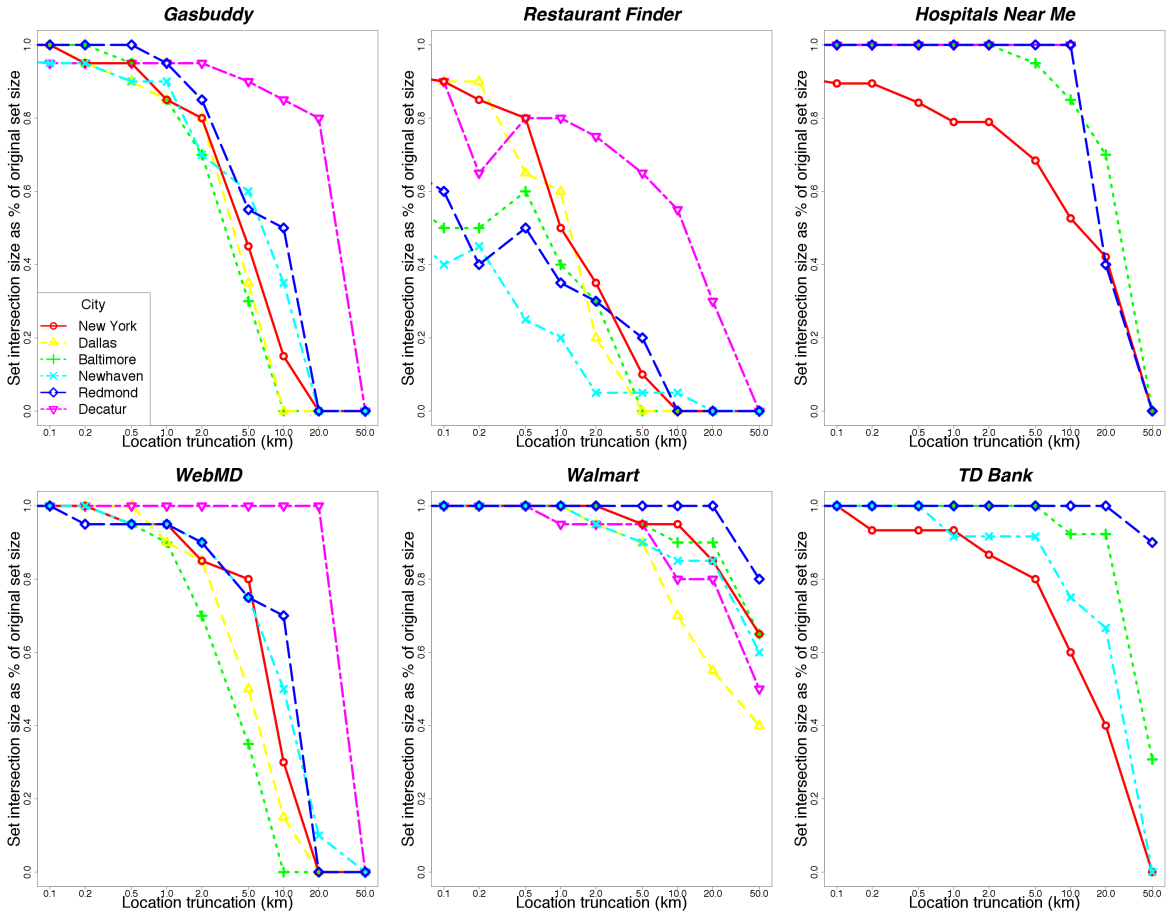
A. Edit Distance

Figure 5 shows how edit distance varies on our location set. Note that Hospitals Near Me and TD Bank do not have data for cities for which the app did not generate enough output to calculate a significant result. For example, there are no TD banks in Dallas. In some cases, TD Bank and Hospitals Near Me do generate data, but they generate a very small set of results (Hospitals Near Me only returns hospitals within 50 miles, for example). Thus, edit distance appears to go down at large truncation values because in less populous locations they output smaller sets of locations.

The plots show that in most cases, the output lists change to some degree even at the smallest truncation level. Moreover, in two cases (Gasbuddy and Restaurant Finder), the edit distance quickly reaches its maximum value (all items in the list change) in several locales. WebMD is similar, though it plateaus somehwat later. The remaining three apps, in contrast, show a generally steady progression of edit distance versus truncation amount.

The reason for these trends is density of objects– Gasbuddy, Restaurant Finder, and WebMD all return lists of items that can commonly be found everywhere, whereas there may be only a few hospitals, Walmarts, or TD Bank locations. Indeed, looking at Gasbuddy, we see that edit distance plateaus quickest for New York, the most populous locale, and slowest for Decatur, the least populous locale.

One problem with the edit distance metric is that even insignificant reordering of the output list adds to the distance—yet users likely will not care about the exact ordering as long as relevant results appear within the first few items of the nominal list. Additionally, edit distance does not have a clear physical interpretation, nor does it seem to correspond to any typical tasks a user might want to perform. Thus, while it provides insight into app behavior under truncation, we think that edit distance is not the best metric of utility.



(a) Graphs of median set intersection size versus truncation amount. Lower set intersection size indicates lower utility.

	New York	Dallas	Baltimore	New Haven	Redmond	Decatur
Gasbuddy	5	5	2	2	5	50
Restaurant Finder	1	0.5	0.1	0.1	0.1	0.2
Walmart	50	10	50	50	·	50
TD Bank	10	·	50	20	·	·
WebMD	10	5	2	10	10	50
Hospitals Near Me	5	·	20	·	20	·

(b) Largest truncation amount (km) before which the median set intersection size is lower than 80%.

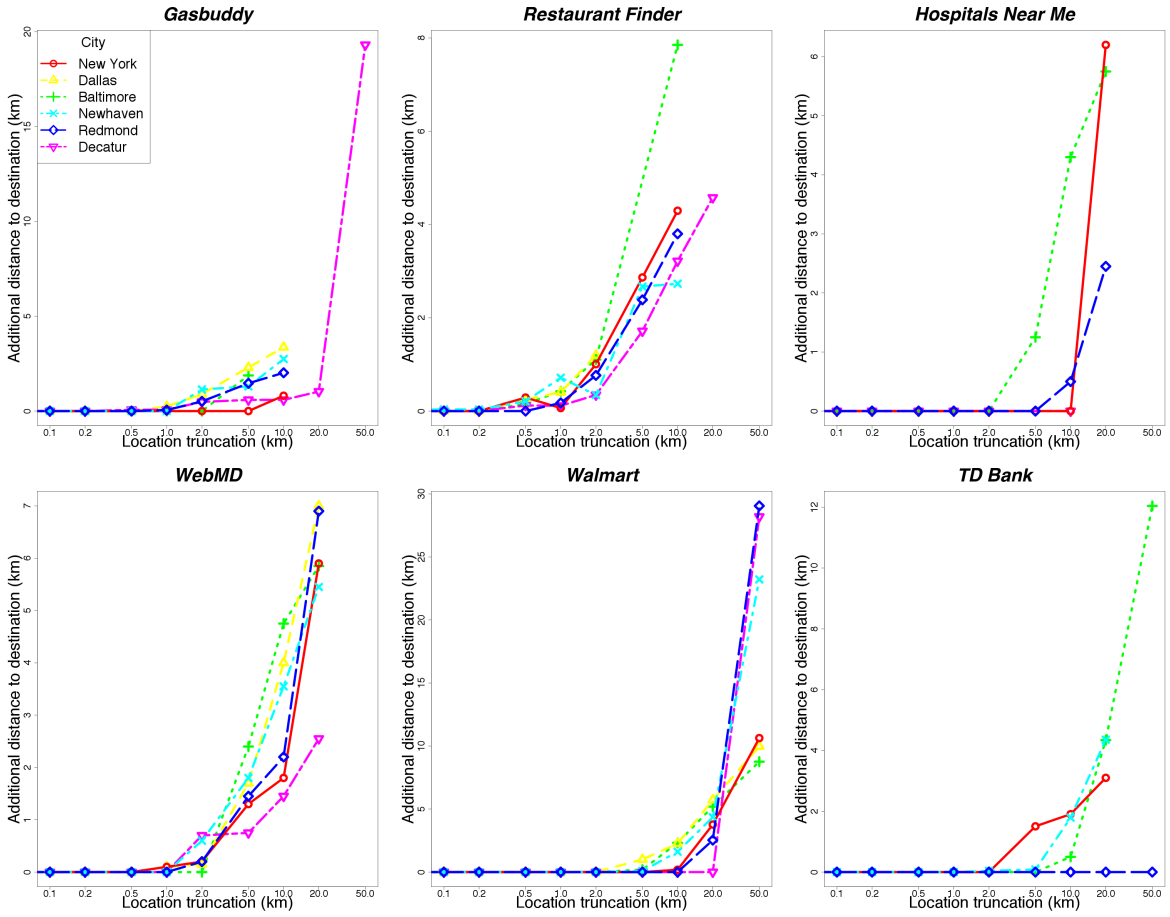
Figure 6: Set intersection size results.

B. Set Intersection

Figure 6a shows the results for the set intersection size metric, with the y-axis the percentage of the set intersection size compared to the original set size. Figure 6b shows the truncation amount after which the set intersection percentage drops below 80% of the original value. More precisely, let $I(s)$ be the median intersection set size under truncation s, defined as $|L \cap L'|/|L|$ where L is the list without truncation and L' is the list with truncation s. Then the chart lists the s_i such that $I(s_i) \leq 0.8$ and $I(s_{i+1}) > 0.8$, where s_j is the sequence of location truncation amounts.

We observe that apps measuring less densely distributed locations can be truncated more aggressively than those measuring more densely distributed locations. For example, we see that Gasbuddy and Restaurant Finder, which both measure densely distributed objects, can sustain some truncation, but their utility drops off much faster than the other apps such as Hospitals Near Me. In the table, we see this again in that Gasbuddy and Restaurant Finder drop below a set intersection size of 80% at a lower truncation amount compared to Hospitals Near Me.

One slightly surprising trend in the table is that for Restaurant Finder, location can be truncated more in New York than in other location, whereas we would expect the



(a) Graph of median additional distance versus truncation amount. Higher additional distance implies lower utility.

	New York	Dallas	Baltimore	New Haven	Redmond	Decatur
Gasbuddy	·	5	5	2	5	20
Restaurant Finder	2	2	2	5	5	5
Walmart	20	5	10	10	20	50
TD Bank	5	·	20	10	·	·
WebMD	5	5	5	5	5	10
Hospitals Near Me	20	·	5	·	20	·

(b) Largest truncation amount (km) before median additional distance exceeds 1 km.

Figure 7: Additional distance results.

opposite due to its high density of restaurants. Looking into the results in detail, we discovered this occurred because of the large radius (20 km) we chose for New York: Restaurants are typically clustered in the city center and we chose locations randomly within the radius, so a large radius reduced restaurant density.

C. Additional Distance

Figure 7a shows the results for the additional distance metric applied to our subject apps. Figure 7b shows, for each city and each app, the last truncation amount before median additional distance degrades beyond 1 km. In other words, if we let $\Delta(s)$ be the median additional distance under truncation s, the chart lists the s_i such that $\Delta(s_i) \leq 1\ km$ and $\Delta(s_{i+1}) > 1\ km$.

Note that the additional distance is undefined if a location truncation produces a new closest location that was not present in the original output list, as in this case we cannot compute how far the new closest location is from the current location. In the charts, undefined additional distances are omitted, as thus some lines end before reaching the maximum truncation. This also limits the number of points on which we can compute the cutoff. For example, the Gasbuddy app never causes the user to go more than 1 km

out of the way in New York, so it is listed as "·" in Figure 7b.

From these plots, we see that under the additional distance metric, there is typically no change in app utility up to 1 or 2 km truncation. Notice that this is quite different behavior than under the edit distance metric, where the edit distance is nonzero even with small amounts of location truncation.

In most cases, the lack of change in additional distance is meaningful, but in a few cases it only reveals the low density of located objects. In particular, in Redmond, the TD Bank app can sustain location truncation up to 50 km without any change, but this occurs because the first result on the list is over 100 km away. This also happens in the Hospitals Near Me app, as there are relatively few hospitals in any area.

Looking at Figure 7b, we see that if a user is willing to go 1 km out of their way, then in most locations apps can sustain truncation amounts of 5 km or more. However, as with edit distance the effect is highly dependent on object density, e.g., Gasbuddy and Restaurant Finder are limited to 5 km truncation (except in Decatur, which has fewer gas stations), whereas other apps can be truncated to higher levels.

D. Discussion

Each of the three metrics applies in a different situations: edit distance for when an exact order is desired, set intersection for when a user wants to find any number of locations close by, and additional distance for when a user wants to find a nearest location to visit. The general trend for all three are similar: the metrics are much more forgiving in less dense areas, which makes intuitive sense. In terms of absolute values of truncation that can be sustained, we see that edit distance is the least permissive, as very small changes have large effects. (We did not include a table like Figure 7b or 6b for edit distance because we have no intuition of what a reasonable cutoff would be.) The next most restrictive metric is set intersection size, and the most permissive metric is additional distance.

VI. Related Work

There is a large body of work on increasing location privacy, though we are aware of no other work that directly measures the resulting utility of mobile Android apps.

The Caché system [17] caches and prefetches content from a server, obfuscating the user's location at the cost of potentially stale content and higher bandwidth. Additionally, application writers must specify rules for caching up front. The system caches data for a set of regions, quantizing the user's location as in our approach. The authors note that app utility will be impacted by their technique, but they do not measure it explicitly.

Hornyack et al. [18] study apps that use sensitive information (such as location), and present AppFence, a system that can fuzz inputs to apps. While the AppFence authors note the effects of fuzzing app inputs on usability, they do not study it systematically using any particular metric.

Shokri et al. [2] present a systematic approach to quantifying Location Privacy Preserving Mechanisms (LPPMs). They also describe an meter on the user's devise that continuously informs the user of their privacy. In our work, we implemented a simple truncation-based LPPM (corresponding to a technique they call *precision reduction*), and use this to study utility as a function of the degree of truncation. As future work, we may consider evaluating the utility of other policies from their system.

In follow up work, Shokri et al. [3] present an optimal strategy to prevent localization attacks. Their analysis formulates location privacy as a Bayesian Stackelberg game, where a user chooses a location cloaking strategy without knowing their potential adversary. While their analysis considers service quality, they use metrics that do not clearly map to utility, such as Euclidean distance from the true location (rather than the actual effect of the change on the service).

Our study focuses on privacy for a single user at a stationary point. Allowing the app to collect traces of data reveals much more information [9], [19]. Much of the existing work on location privacy [1], [4], [5], [6] focuses on k-anonymity: if a user is making a query to a location based service, they can only be identified to be within a set of k potential users. One popular technique is to use mix zones [1]: once a user enters a designated area their location information becomes "mixed" with others in that same area. This technique requires a trusted middleware layer (to properly mix location data) and requires these mix zones to be defined. In contrast, location truncation can be done locally on a mobile device.

None of these previous approaches study the impact of location privacy on app utility. The work that comes closest is by Shokri et al. [3], but while that framework considers utility as an element of their models, it is not directly measured on apps. Our work is complementary: we focus not on optimal obfuscation techniques, but rather we fix an obfuscation strategy and study how utility changes under that technique. As future work, we intend to couple our empirical utility functions with the theoretical models presented by Shokri et al. Doing so would allow us to determine bounds for k and allow us to study how the optimal strategies presented by Shokri et al. are affected.

VII. Conclusion

We presented an empirical study of how location truncation affects mobile app utility. We began by examining how Android apps use location. Across the apps we looked at, we found that the second most common pattern is using the current location to list various sorts of nearby objects. The most common pattern, location-targeted ads, is not amenable to evaluating utility. We then used Dr. Android and Mr. Hide to implement CloakDroid, a tool that modifies existing apps' bytecode to use truncated location information. We designed an experiment in which we measured the effect of a range

of location truncations (from 0 km to 50 km) on 60 points randomly chosen from various locales (from population 6,000 Decatur, TX to population 8.2 million New York, NY). We identified three metrics that approximate app utility: edit distance, set intersection size, and additional distance. We found that, under these metrics, the factor that most determines the utility–truncation tradeoff is the density of objects being returned by the app, and that in many cases, location can be truncated significantly without losing much utility. To our knowledge, our work provides the first end-to-end evaluation of how location truncation affects Android app utility. As subject of future work, we plan to apply CloakDroid to a larger number of apps and test the metrics using user studies.

REFERENCES

[1] A. R. Beresford and F. Stajano, "Mix zones: User privacy in location-aware services," in *IEEE Conference on Pervasive Computing and Communications Workshops (PERCOMW)*, 2004, pp. 127–.

[2] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux, "Quantifying location privacy," in *IEEE Symposium on Security and Privacy (SP)*, 2011, pp. 247–262.

[3] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. Le Boudec, "Protecting location privacy: optimal strategy against localization attacks," in *2012 ACM conference on Computer and Communications Security (CCS)*, 2012, pp. 617–627.

[4] C. Bettini, X. S. Wang, and S. Jajodia, "Protecting privacy against location-based personal identification," in *VDLB International Conference on Secure Data Management (SDM)*, 2005, pp. 185–199.

[5] B. Hoh and M. Gruteser, "Protecting location privacy through path confusion," in *Security and Privacy for Emerging Areas in Communications Networks*, 2005, pp. 194–205.

[6] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2003, pp. 31–42.

[7] A. B. Brush, J. Krumm, and J. Scott, "Exploring end user preferences for location obfuscation, location-based services, and the value of location," in *ACM International Conference on Ubiquitous Computing (Ubicomp)*, 2010, pp. 95–104.

[8] J. Krumm, "A survey of computational location privacy," *Personal Ubiquitous Comput.*, vol. 13, no. 6, pp. 391–399, Aug. 2009.

[9] M. Gruteser and B. Hoh, "On the anonymity of periodic location samples," in *International Conference on Security in Pervasive Computing (SPC)*, 2005, pp. 179–192.

[10] J. Krumm, "Inference attacks on location tracks," in *International Conference on Pervasive Computing (PERVASIVE)*, 2007, pp. 127–143.

[11] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications," in *ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012, pp. 3–14.

[12] Google. (2013, Feb.) Bytecode for the dalvik vm. [Online]. Available: http://source.android.com/tech/dalvik/dalvik-bytecode.html

[13] National Geospatial-Intelligence Agency. (2013, Feb.) Length of degree calculator. [Online]. Available: http://msi.nga.mil/MSISiteContent/StaticFiles/Calculators/degree.html

[14] R. H. Rapp, "Geometric geodesy: Part i," *Lecture Notes*, 1991.

[15] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[16] J. Jeon and J. S. Foster, "Troyd: Integration Testing for Android," Department of Computer Science, University of Maryland, College Park, Tech. Rep. CS-TR-5013, August 2012.

[17] S. Amini, J. Lindqvist, J. I. Hong, M. Mou, R. Raheja, J. Lin, N. Sadeh, and E. Tochb, "Caché caching location-enhanced content to improve user privacy," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 14, no. 3, pp. 19–21, Dec. 2010.

[18] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *ACM Conference on Computer and Communications Security (CCS)*, 2011, pp. 639–652.

[19] P. Golle and K. Partridge, "On the anonymity of home/work location pairs," in *International Conference on Pervasive Computing (PERVASIVE)*, 2009, pp. 390–397.