Caché: Caching Location-Enhanced Content to Improve User Privacy

Shahriyar Amini
Carnegie Mellon University
Pittsburgh, PA, USA
shahriyar@cmu.edu

Janne Lindqvist
Carnegie Mellon University
Pittsburgh, PA, USA
jklindqv@cs.cmu.edu

Jason Hong
Carnegie Mellon University
Pittsburgh, PA, USA
jasonh@cs.cmu.edu

Jialiu Lin
Carnegie Mellon University
Pittsburgh, PA, USA
jialiul@cs.cmu.edu

Eran Toch
Tel Aviv University
Ramat Aviv, Tel Aviv, Isreal
erant@post.tau.ac.il

Norman Sadeh
Carnegie Mellon University
Pittsburgh, PA, USA
sadeh@cs.cmu.edu

ABSTRACT

We present the design, implementation, and evaluation of Caché, a system that offers location privacy for certain classes of location-based applications. The core idea in Caché is to periodically pre-fetch potentially useful location-enhanced content well in advance. Applications then retrieve content from a local cache on the mobile device when it is needed. This approach allows an end-user to make use of location-enhanced content while only revealing to third-party content providers a large geographic region rather than a precise location. In this paper, we present an analysis that examines tradeoffs in terms of storage, bandwidth, and freshness of data. We then discuss the design and implementation of an Android service embodying these ideas. Finally, we provide two evaluations of Caché. One measures the performance of our approach with respect to privacy and mobile content availability using real-world mobility traces. The other focuses on our experiences using Caché to enhance user privacy in three open source Android applications.

Categories and Subject Descriptors

H.5.m [**Information interfaces and presentation**]: Miscellaneous; K.4.1 [**Public Policy Issues**]: Privacy

General Terms

Experimentation, Security, Human Factors

Keywords

Location privacy, location-enhanced content, location-aware applications, disconnected operation, caching

MobiSys'11, June 28–July 1, 2011, Bethesda, Maryland, USA.

1. INTRODUCTION

In recent years, location-aware devices, such as mobile phones with GPS, have gained mainstream popularity. This trend has led to a rapid increase in location-based services [34, 42]. Examples of such location-based services include support for finding gas stations[1] and stores[2] with the lowest prices, finding friends, and notifying friends when you arrive at a location. Other examples include location-based games[3] as well as location-enhanced micro-blogging.

A key challenge to widespread adoption of location-based services, however, is privacy [19]. One problem is the perception of privacy: people have expressed many concerns about being tracked by friends and by third parties. Location privacy concerns also tend to attract negative media coverage, further hindering the spread of location-based services. Another problem is actual privacy: end-users may be unaware of the privacy implications of location-based technologies [3, 4], and end up unintentionally sharing more information than they realize.

To address this problem, we present Caché, a generalizable approach for a class of location-based services that enables users to enjoy the benefits of those services while minimizing the associated privacy concerns. Caché takes a well-explored idea from distributed systems, namely caching, and applies it in the context of privacy. Caché has two core ideas: (1) location-enhanced content can be periodically pre-fetched in large geographic blocks onto a device before it is actually needed, for areas that a person will likely be in, and (2) the content can be accessed locally on a device when it is actually needed, without relying on any networked services outside of the device. Thus, rather than sharing current location on each request for information, the user only needs to share general geographic region hours, days, or even weeks before the desired content is needed.

There are four steps in using Caché. Let us consider a scenario in which a restaurant finder application is developed to use Caché. First, at design time, the developer provides some hints as how to download the content (e.g., URL, content update interval). Second, the user installs the Caché-enabled application and selects the regions for which restaurant POIs should be downloaded. Third, Caché down-

[1]GasBag, http://www.jam-code.com
[2]ShopSavvy, http://www.biggu.com
[3]JOYity, http://www.androidapps.com/t/joyity

loads and updates the content based on the developer specified update rate, at a favorable time (e.g., when the device is powered and a WiFi connection is present). Finally, when the application requires content, it retrieves the content from Caché, rather than make a live query.

Assuming that the user's current location is determined locally on the device (as is the case with GPS, PlaceLab [15, 24], POLS [35], and SkyHook's autonomous mode), the user can still make use of location-enhanced content, while content providers that offer maps, restaurant guides, bus schedules, and other location-enhanced content are only aware of the user's general area of interest.

In past work [2], we presented a brief feasibility analysis and outlined a possible implementation of Caché. In this paper, we present the design, implementation and evaluation of Caché. More specifically, this paper makes the following research contributions:

- A feasibility analysis of caching for privacy, including a taxonomy of location-based data types, and a discussion of tradeoffs with respect to freshness of data, storage, and bandwidth requirements
- A system architecture that through pre-fetching enables the use of location-enhanced content while also supporting user privacy
- A reference implementation of our approach
- A performance analysis that demonstrates the benefits of caching, specifically, the increase in privacy with respect to the increase in bandwidth and storage usage, evaluated through the use of two real-world mobility trace databases
- Our experiences using Caché to improve privacy in three open source Android applications

The rest of the paper is organized as follows. Section 2 presents feasibility analyses of a cache-based privacy solution. We discuss our design requirements in Section 3 and the system architecture in Section 4. We present the evaluation in Section 5, and end with discussion, related work, and conclusion in Sections 6, 7, and 8.

2. FEASIBILITY ANALYSIS OF CONTENT CACHING

In this section, we provide an analysis of some of the technical challenges in caching location-enhanced content. More specifically, we address the following:

- Location privacy model
- Cache hits and cache misses
- Data freshness
- Data consistency
- Estimated storage requirements
- Estimated bandwidth requirements

Note that in this section, we focus mainly on the technical issues involved. There are many issues beyond the scope of this paper, for example, how caching might impact advertising on web sites, not to mention the various legal issues involved in caching a great deal of content.

2.1 Location Privacy in Caché

Currently, there exist numerous ways to acquire location-enhanced content, however, each presents a different location privacy trade-off. One model for acquiring location-enhanced content is to make a live request to a content provider. Such a request would result in the user providing some information about her current interests as well as her current location. The worst-case scenario consists of a third party knowing when and where the user will be present at all times.

On the other side of the spectrum, the user can purchase content prior to usage. One example would be purchasing a copy of Microsoft MapPoint, which comes pre-loaded with maps and points of interests. Although the user's requests for content are kept private as they are fielded by the purchased content, the data may become stale and may lack future updates. As a result, this model does not work well for content that changes frequently. There is also a middle ground where the user can purchase content in bulk, and update the content whenever updates are available from the content provider. Purchasing a GPS device and updating the maps and points of interests is an example of this model. However, this method also does not handle content that changes frequently such as social events and movie theater schedules. Unless the user's device is connected and set to update automatically, it is unknown when the user's content will be updated, if at all.

Caché resides in the middle of the spectrum. Data is stored on the user's device for the user's regions of interest and is also updated to reflect changes. As such, the user has two points of interaction with Caché. First, by pre-fetching content for a geographic area, the user signals that she is interested in that area, but not specifically when and where she is in the area. Second, the user implicitly requests content from Caché when interacting with Caché-enabled applications. Assuming that location is determined locally, then there is not any information shared outside of the mobile device. An obvious issue here is that this pre-fetching approach only works if we can pre-fetch useful content for the correct geographical regions in advance, and if the content does not change very often.

2.2 Cache Hits and Cache Misses

While using Caché, there are three possible outcomes when looking for content near one's current location: (1) the content is cached and up-to-date, (2) the content is cached but is out-of-date, and (3) the content is not cached. The first case is a positive outcome, and with Caché we seek to maximize this outcome.

The second case, out-of-date content, means that content is available but may not be fresh. In some situations, this is acceptable. For example, traffic information is not very useful days after the fact, but maps can still be useful even if they are a few years out-of-date. A related problem here is that users will likely be unaware of the freshness of data until it is requested. This can have an adverse effect on the user experience, as in the case of stale traffic data.

The third case, a cache miss, could be caused by the user not having cached content for a specific area of interest, or the user moving outside the boundary of cached content. In this case, the choices are to display no relevant results or to download the content from a service provider on demand, at the potential cost of some privacy and slower performance.

Table 1: A comparison of bandwidth and storage required for different types of content for the Pittsburgh metropolitan area. In our application, we focus only on data updated with at most a daily frequency.

Update Rate	Data Type	Size†	Time to Download‡
Real-Time (STTL)	traffic flow, parking spots e.g., Loopt, PeopleFinder, Reno, Bustle	–	–
Daily	**weather forecasts**, social events, coupons e.g., Dede [18]	<1 MB	<1 min
Weekly	**movie/theatre schedules**, advertisements, crime rates e.g., Yelp!, GeoNotes, PlaceIts, PlaceMail	1.4 MB	<1 min
Monthy	restaurant guides, **bus schedules**, geocaches e.g., Wikipedia (geotagged pages)	4.2 MB	2.8 min
Yearly	**maps**, points of interest, tour guides, store locators e.g., Google Maps, Starbucks, Wal-Mart	6.4 MB	4.3 min

†Storage estimates are provided for the bolded content type.

‡Download times are estimated for a 200 kbps connection for the bolded content type.

2.3 Data Freshness

In this section, we analyze the feasibility of the Caché approach with respect to the freshness of cached data. We define two high-level categories to distinguish between the update frequency requirement of data types, namely, Short Time to Live (STTL) and Long Time to Live (LTTL) data types. STTL refers to data that requires updating in real-time or close to real-time, for example traffic information. LTTL data refers to data that can be updated less often, on the order of days or greater.

Table 1 shows examples of STTL and LTTL location-based data types. As noted earlier, there are many kinds of applications that require real-time updates, in particular those making use of synchronous communication. If the user has a stable network connection, she can choose to keep STTL data fresh by allowing for more frequent updates. However, this approach depends on the time it takes for STTL content to become stale, requires a network connection, and potentially costs the user some privacy. There are also many kinds of data types that do not require immediate and constant updates. Some data types only need to be updated once every day, such as weather conditions, social events, and coupons. Other data types can be updated on a weekly, monthly, or yearly basis, without compromising the quality of the content. For example, a user might refresh stale LTTL content overnight, and make use of it the following day.

To evaluate the feasibility of the proposed solution, we analyzed location-enhanced content downloaded daily from May 2009 to October 2009. We downloaded data for weather, social events, bus schedules, restaurant points of interests from MSN and Yelp!, and Google map tiles. The rationale for our analysis was to assess whether the selected content type could be cached overnight and still provide fresh accurate data when mobile and disconnected. We studied each data type with respect to the percentage of data added, removed, and modified daily (See Table 2). Based on our findings, we have concluded that it is feasible to download the aforementioned data types ahead of time to preserve user privacy. We describe our findings for each data type below.

Weather: We downloaded weather data everyday using Google's weather API, which offers weather information daily for four consecutive days, starting with the present day. For each weather caching instance, the data for the previous day's weather becomes stale and has to be removed. From any day, the weather condition for the next three days carry over. As a result, there is 25% weather data added daily, and 25% removed. Based on the data collected, approximately 67% of the weather data changed on average. We performed our comparison considering minor changes in the temperature or weather conditions as complete invalidation of prior cached data. If one were to consider minor changes in temperature such as from highs of 72 degrees Fahrenheit to 73 degrees Fahrenheit insignificant, then the modified data percentage would be lower. Caching weather conditions the night before use is a feasible solution for being aware of present day's weather conditions. The stored weather data for the next three days would allow for some degree of information regarding upcoming weather.

Social Events: We downloaded events for Pittsburgh and surrounding areas from Zvents.com, a major database that stores upcoming events by aggregating and enforcing content standards from various sources. We centered our search at Pittsburgh with a radius of 75 miles and downloaded events every day for five months. The site consistently offers approximately 2000 events in the search area. Based on our data, we found less than 6% was added and removed daily, with about 12% modified daily. Thus, over 80% of data was fresh on a day to day basis.

Points of Interests: We downloaded restaurant data from Yelp! and MSN for Pittsburgh. The add, remove, and modification percentages for Yelp! were all below 1%. The MSN search modification percentage was below 2% with add and remove percentages of approximately 7%. These percentages are low enough that the user would have a substantially fresh cache for all restaurant data for Pittsburgh. Based on these obtained results, we conjecture that other point of interest data, such as gas stations, store locations, libraries, and transportation stations, would behave in a similar manner as physical places are relatively slow changing.

Bus Schedules: We downloaded Pittsburgh bus schedules from the Pittsburgh Port Authority. The bus schedule content was downloaded as html pages and parsed. Over the five month period, there was no occurrence of schedules being added or removed. We noted a 0.15% change in the schedules, reflecting minor changes in departure and arrival times over the five month period.

Maps: We downloaded map tiles from Google Maps for the Pittsburgh metropolitan area and surrounding suburbs.

Data Type	Added %	Removed %	Modified %
Weather	25.00	25.00	67.26
Events	5.28	5.35	11.75
Yelp POI	0.15	0.06	0.04
MSN POI	6.69	6.80	1.43
Bus Schedule	0.00	0.00	0.15
Map Tiles	0.00	0.00	0.00

Table 2: Average daily percentage of change for added, removed, and modified data for various data types. The results are based on downloading the respective content daily for five months starting in late May 2009. The added and removed columns refer to percentage of new entries added to or removed from the cache each day, respectively. The modified column refers to the percentage of entries that remained in the cache but required changes to update data.

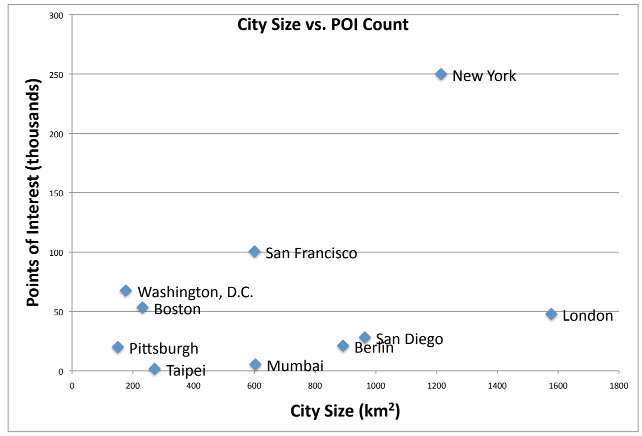

Figure 1: POI data for selected metropolitan areas from Google Maps. In the case of foreign cities, notably Taipei and Mumbai, Google does not have a comparable amount of data.

At a zoom level of approximately 3.70 meters per pixel[4], all of the 226 downloaded map tiles required only 6.4 MB of storage. We found that over a period of more than five months, there was zero change in the Pittsburgh map tiles. This was verified by accounting for additional map tiles that had to be downloaded, map tiles that had to be removed, and whether an md5 hash of each individual map tile changed.

Based on the presented results for weather, events, points of interests, bus schedules, and maps, we conclude that it is feasible to cache data overnight to field user queries during daily use from the local cache. We notice that weather conditions change the most on a daily basis. However, considering automated caching of weather information every night, the user will have reasonably fresh content for the morning. Further, the cached content indicates some level of information regarding the weather, even if marginally stale.

2.4 Data Consistency

Caché has a simple consistency model, in that it only reads data from the web, and never writes data back. Thus, Caché considers data on web sites to be canonical, and data stored on a mobile device to be a soft copy that can always be overwritten. As such, the main data consistency problem resides in maintaining data freshness, as discussed in the previous section.

2.5 Estimated Storage Requirements

In this section, we provide an analysis of the estimated storage requirements for caching location-based data types for a typical city. We start by examining map data for streets. As previously mentioned map tiles were downloaded for the city of Pittsburgh and surrounding suburbs. In total, the map tiles required only 6.4 MB of storage. We repeated the analysis for New York City and found that the entire city could be stored in approximately 65 MB at 3.7 meters per pixel. Considering that entry-level portable music players come with 4 GB of disk space, storing map data on a mobile device for a number of cities is quite feasible.

We continue by estimating the storage requirements for points of interest, using restaurants as a starting point. There are approximately 20,000 restaurants in the city of New York[5]. With our current implementation, which caches information regarding the restaurant's name, street address, and GPS coordinates, all such entries can be cached in about 10 MB. The amount of storage increases based on the number of categories for which the user maintains a cache.

We performed a separate analysis to estimate storage requirements for POIs. We used New York City as a feasible upper bound for the amount of necessary content for a given city. Using Microsoft MapPoint as the source of points of interest data, we obtained a total of approximately 76,000 points of interest by searching a 35 mile radius [27]. Given the most basic record of name, point of interest type, street address, latitude, longitude and a brief description, all such data points can be stored in less than 33 MB.

Figure 1 shows the number of points of interests for various cities as obtained through queries on Google Local. In this approach, New York City has about 250,000 points of interest, requiring about 100 MB of storage. Thus, we have roughly 65 MB of map tile data and roughly 100 MB of POIs for a very large city. Given this analysis, basic location-enhanced content can be easily stored on modern mobile devices, even if the number of points of interest were increased by two orders of magnitude.

Note that our estimates for points of interest focus on text rather than images, audio, or video. Storing rich media for New York City would require more space than a mobile user could afford. Although we have not yet explored multimedia content, we have not ruled out the possibility of storing content by reducing content fidelity [30], or reducing the size of the cached region, i.e., caching on a ZIP code or neighborhood level. For streaming multimedia content, the user would have to rely on other methods of maintaining privacy.

2.6 Estimated Bandwidth Requirements

Bandwidth is a potential challenge since it could take an unreasonable amount of time to download content in some cases. In this paper, we estimate the time to download con-

[4]Corresponding to a zoom level of 15 for the Google Maps API at a latitude of 40° north.

[5]NYC Department of Health
http://www.nyc.gov/html/doh/html/rii/index.shtml

tent based on a conservative connection speed of 200 kbps, which is the FCC required minimum for a connection to be considered high-speed Internet access. Presently, much higher transmission speeds are available in the United States [41]. Assuming the worst case of refreshing all map tiles and points of interest every day, pre-fetching for New York City would take approximately 2 hours to complete on a 200 kbps connection. This would be enough time to perform a complete download overnight, such that the content could be used the following day.

The challenge comes when more content is needed. In the previous section, we observed that mobile devices could easily store gigabytes of data. However, the approach is limited by the amount of content that could be downloaded in a reasonable amount of time. If we assume we have six hours to download content at 200 kbps, then this yields roughly 530 MB of content refreshed per day. Again, this is sufficient for simple forms of location-enhanced content, including much of the data types in Table 1, but not enough for multimedia.

3. DESIGN REQUIREMENTS

We briefly outline the design requirements for Caché in this section. For deployability, it is preferable that a privacy-preserving approach relies only on the mobile device, and minimizes reliance on infrastructure beyond content itself. Service providers might not have incentives to preserve users' privacy, which is the motivation for our work, and further, relying on the mobile device simplifies trust assumptions. Furthermore, we aim to minimize required user interaction. Preferably, a privacy-preserving approach would be completely transparent to the users and application developers, but as we discuss later, this is not a practical approach. As such, we placed the burden of managing privacy on application developers, and offer support to simplify the task of making applications privacy sensitive. Note that by application developer we are referring to a person who would be developing the application client for the user's mobile device. However, there is no reason why the client-side and server-side developer could not be the same person.

Threat Model: Our objective is to minimize the information flow towards location-based service (LBS) providers, or anyone accessing their logs or observing the traffic on the way. Further, we explicitly desire that the real-time and precise location of Caché users cannot be accurately determined. However, we do not protect against a local eavesdropper such as the network access-point, which can directly observe that the user is downloading traffic related to multiple locations. A detailed applicable threat model for the problem space has been throughly discussed before, e.g., by Gruteser and Grunwald [12]. We emphasize that the threat is not only that somebody discovers the location and regular patterns of users movements through LBSs or logs of intermediate servers. These logs can also be used to reveal the sender of a message, if the attacker knows that a location belongs to a user and discovers that the user was in that location at a particular time.

4. CACHÉ SYSTEM ARCHITECTURE

The Caché architecture depicted in Figure 2 is similar to that of an (non-transparent) Internet proxy requiring registration. When requesting location-enhanced content, instead of querying the service provider directly, the application submits its request to Caché. Caché processes the query and provides the application with the requested content.

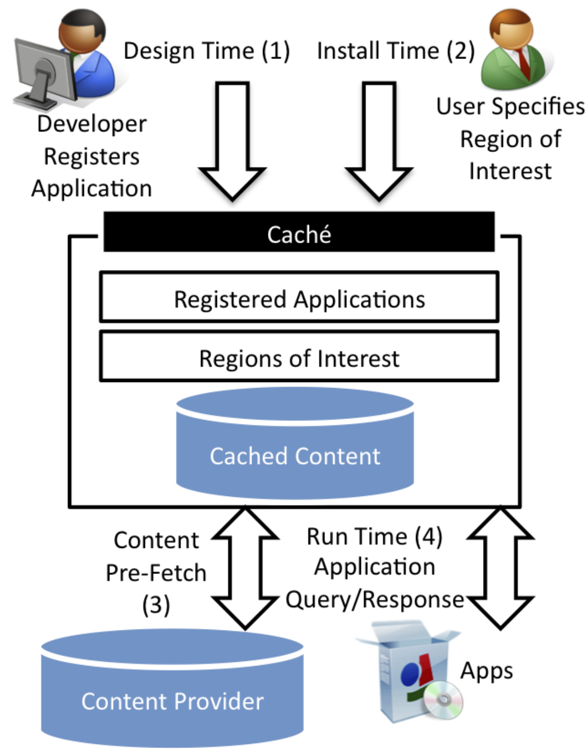

Figure 2: Once the developer has registered an application (1) and the user has specified the regions for which content should be cached (2), Caché requests and stores the content (3) for future use by the application (4).

The steps towards using Caché are as follows. The developer registers the application with Caché. The registration is a simple declaration of the application's content needs and the content request format. After installation and prior to initial use of the application, the user specifies the region for which content should be downloaded by specifying an address such as home or work, or a ZIP code. Caché then downloads the content for the specified region using as many requests to the content provider as necessary to cover the entire region. The content download optimizes cost and energy by downloading content only when the device is plugged in and a WiFi connection is available. At runtime, all application requests are forwarded to Caché, which takes a best effort approach to provide the relevant content.

4.1 Space Discretization

In this section, we describe how and why Caché uses space discretization with content downloading. Location-enhanced queries use latitude and longitude to describe geographical regions. However, it is not possible to download content for every latitude and longitude combination. To address this problem, Caché decomposes a geographic region of interest into a grid of cells. The grid consists of same sized rectangular cells. The size of the rectangular cells are defined by the developer at the application registration stage. This requires the developer to have a notion of how densely

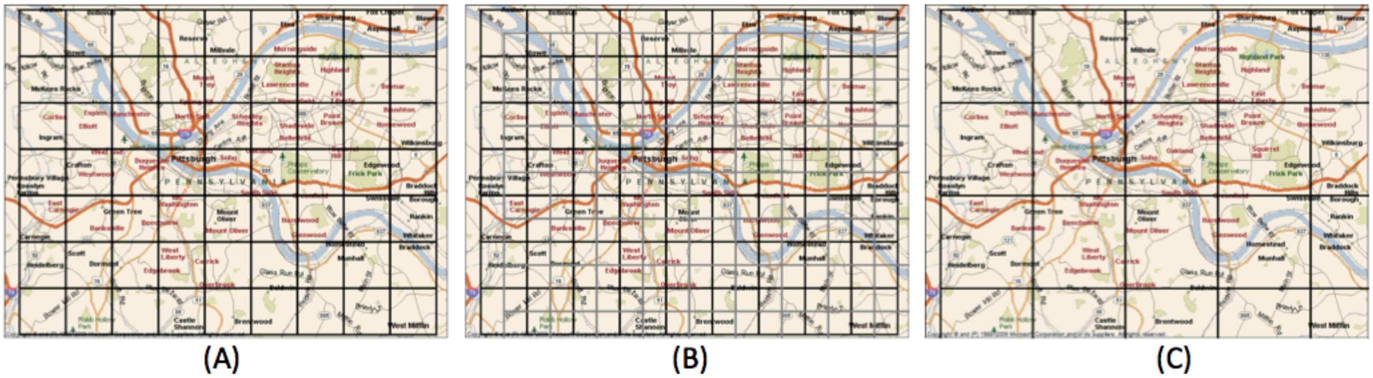

Figure 3: The user's region of interest is discretized in space based at the granularity defined by the application developer. Caché makes queries for each cell. When the application makes a content request, the results for the nearest cell covering the query region are returned. (A) shows a level 0 grid, which has the smallest cell size. (B) shows the same level grid with a grid overlay placed. (C) shows a level 1 grid. Cell height and width are doubled at each level.

the content is packed in terms of physical geographical space. An example grid is present in Figure 3(A). Our space discretization is based on the work by Krumm [21], which uses a grid for location obfuscation. We use the grid as a basis to download content with two optional modifications: overlay and hierarchy.

Let's consider the case where the user's request for content falls in the corner of a cell. The content retrieved from that particular cell may not be the most relevant. We use a *grid overlay* to tackle this problem (see Figure 3(B)). Overlay refers to a secondary grid with cells of the same size as the original that are shifted by a quarter of a cell. This way, requests that fall at a corner of a grid cell can be fulfilled with one of the overlay cells, instead of an original grid cell. Using an overlay effectively doubles the amount of content that has to be pre-fetched beforehand and also considered during application content request. The grid overlay is an optional feature, which is selected by the developer at application registration time.

The other case to consider is when a request region encompasses a large area. If a single grid with same sized cells were to be used, then the request could potentially cover multiple cells. However, Caché does not have any information regarding the data semantics that it stores. As a result, it will not be able to combine data from multiple cells to service a single request. We use *grid hierarchy* to service larger request regions (see Figure 3(A) and 3(C) for a level 0 and level 1 grid, respectively). Grid hierarchy refers to additional levels of cells that grow to cover the user's entire region of interest. We refer to the grid with the smallest granularity cells (defined by the application developer) as the level 0 grid. At each level, the width and the height of the cells are doubled in size until a single cell covers the entire region of interest. The hierarchy option can be selected by the developer at the application registration time. Note that both the overlay and the hierarchy options may be selected.

4.2 Caché Setup and Usage

In this section, we describe what happens at each stage of the Caché application lifecycle: design, installation, content download, and content retrieval.

4.2.1 Application Design

At design time, the developer has to register the application with Caché. Application registration informs Caché of the nature of content requests, request parameters, the finest sized grid cell for which content is downloaded, and whether the optional overlay and hierarchical grids are to be used.

Location-Enhanced Content Request: Some service providers offer programming language-specific APIs to request content (e.g., Google's JavaScript and Bing's C#), however, we focus only on RESTful HTTP requests. Our decision to focus on REST-based requests is due to limited programming language-support on mobile platforms. For instance, not all mobile platforms offer C# support. Nevertheless, application developers can take advantage of REST requests across multiple platforms, e.g., Android, iPhone, Windows Phone 7, and Symbian.

We studied the content request formats of three major content providers: Microsoft Bing Local, Google Local, and Yelp!. We also looked at the content services GeoNames and Google Panoramio used by two open-source Android applications, Mixare and Panoramio, respectively. We chose these two applications as they are popular Android open source applications that also fit the LTTL model. Through the study of these applications and services, we noted general forms of requesting location-enhanced content from service providers. Below is a summary of request formats discovered:

Single Geo Coordinate: Consists of latitude and longitude coordinates. Google Local, Bing, and Yelp! all supported this request format.

Geo Coordinate + Radius: The region described is centered at the geo coordinate and bounded by a circle with the specified radius. Bing and GeoNames (Mixare) both supported this format.

Bounding Box: defined by two Geo Coordinates such as the top right and bottom left coordinates of a rectangular region. Yelp! supports this location format.

Geo Coordinate + Span: a single Geo coordinate and the subtended latitude and longitude degrees, describing a rectangular region. Google Local supports this.

Geo Coordinate Range: a latitude range and a longitude range, approximately defining a rectangular region. Google Panoramio supports this format.

Caché is capable of addressing the requests and storage needs of all the previously mentioned request types. As long as other content providers use similar patterns of requesting content, no changes are required. Nevertheless, Caché can easily be extended to support other content providers by the simple addition of request parsers that map the new provider's request to one of the pre-loaded request types.

Query String: During the application registration step, the developer provides Caché with a query format string describing the content provider's URL and request format. For instance, for a single geo coordinate Yelp! query, the developer would register the application with a format string similar to *http://api.yelp.com/v2/search?term=food&ll=#SLL_LAT#,#SLL_LON#*. *#SLL_LAT#* and *#SLL_LON#* are Caché parameters, specifying where the Single Latitude Longitude (SLL) values should be placed. Quantities such as radius can be specified in various units such as meters, km, and miles (e.g., *#RADIUS_METERS#*, *#RADIUS_KM#*, *#RADIUS_MILES#*). Details for parameter conventions and request types are compiled as a technical report [1].

Non-Numeric Request Parameters: The developer can also add non-numeric parameters to the request. For example, if each query contains a string which could be any of *pizza*, *burger*, or *wings*, the developer can specify a string part of the query which can be mapped to any of the above. Caché would then make requests for all combination of query inputs and grid cells. The developer can also pick an argument that encompasses multiple query values based on domain knowledge (e.g., *restaurants* instead of *pizza*, *burger*, or *wings*). A query string is not always necessary for content request. For instance, bus schedules, weather, and social events may use a textual argument to narrow down a selection, but may be stored without one.

Cell Size, Overlay, and Hierarchy: The developer chooses the size of the grid cells and whether an overlay and/or a grid hierarchy should be used based on content domain knowledge.

Update Rate and Scheduling Priority: During application registration, the developer also provides the update rate and the priority of the content. The update rate is in terms of days. The priority ranges between 0 to 9, as defined in the technical report [1]. The update rate allows the developer to declare where the content belongs in the LTTL spectrum. Caché downloads the content based on the update rate, in increasing priority order.

Content Storage: The developer also selects the content storage format. Caché stores content in its original format in a database as text, a blob (binary representation), or a file. Content stored as a file is stored directly onto the file system with a pointer to its location. The text format allows for optimized compression and covers a universal content formats. Blob gives the developer more freedom, however, requires the developer to deal with the binary representation formatting.

4.2.2 Application Installation

After installing a Caché-enabled application, the user has to specify the region for which content should be downloaded (Figure 2, step 2). We allow for a number of input methods. Specifically, the user can select between the user's current location, an address, a ZIP code, or a city. The user also specifies the radius in miles of the region centered at the aforementioned geographical location. Multiple regions may be entered for content download, for instance, for possible travel destinations or for work or school. The user may also modify the content update rate and priority.

4.2.3 Application Content Download

Content download refers to step 3 of Figure 2. In this section, we present aspects of downloading and storing content, namely, the downloading approach, content storage, and content ranking.

After application registration and specification of content download regions, Caché has all the necessary information to download content. Caché makes requests based on the URL that the developer has provided and the grid that Caché builds over the region of interest. The cells can be downloaded in any order, e.g., sequential or random.

In the case of a hierarchical grid, Caché starts requesting content for the highest level grid first; grid levels are assigned at grid construction time. Since the download takes place over the region specified by the user, one may argue that the center of the region may be an important location to the user, e.g., home or work. However, important locations are masked as we use space discretization. The center location can only be determined to the smallest size grid cell [21]. Also, we encourage the user to enter the region of interest as a larger entity, such as a ZIP code or a city. As a result, the user's home or work is not necessarily at the center of the region. Other possible forms of mitigation are introducing noise to the original user's region, downloading content for a larger region, or using anonymity approaches where all the people wanting content for an area would have the exact same request pattern.

Content Ranking: By ranking we refer to the order in which a content provider presents individual results to a query, e.g., the order of restaurants from Yelp!. Caché can both preserve and also distort content rankings. In cases where there is no need for hierarchical caching and input query strings, Caché preserves the content ranking as selected by the service provider. If the developer has selected cell sizes that poorly describe content density, the ranking could be distorted. For instance, if the developer selects the cell size to be as large as a city, then the POIs would be centered somewhere in the city, whereas the user may be anywhere in the city.

If hierarchical caching is necessary, the content returned may be for a region that is possibly larger than the request region. This may also result in ranking distortion. In cases where an input query is necessary and an overarching query is used, e.g., *restaurants* instead of *pizza, burger, wings*, the more refined options are dropped from the content presented, even though the ranking may be almost equivalent.

4.2.4 Application Content Retrieval

Content retrieval refers to step 4 of Figure 2, where RESTful requests are sent to Caché. How the content is retrieved depends on the complexity of the grid as defined by the developer and the user.

When hierarchical caching is not used, content retrieval is simple. Assuming the request location is somewhere in the grid, the content from the nearest cell in the grid is selected. Note that the content only corresponds to a single cell. As

Caché does not know about content semantics, there is no notion of taking a union or an intersection over the content. Therefore, Caché finds the nearest cell in the database that covers most of the request region, and presents its content to the application. There are no additional changes required in the application as the content is presented in exactly the same format as that from the service provider. Because content is retrieved based on a cell rather than the user's exact position, the content may not be exactly what the user would have retrieved if she were to make a live request.

When hierarchical caching is used, cells are searched in increasing grid level order. The lowest level grid has the smallest cells. The closest cell that covers the entire request is returned. If the nearest cell at level l does not cover the entire query region, Caché moves up a level to $l+1$ to find a cell that covers the entire region. Upon finding the particular cell, Caché returns the corresponding content.

With respect to grid overlays, the concept remains the same, except there are double the number of cells at each level. The overlay allows for higher content accuracy at each level. Further, the overlay might allow Caché to satisfy the request region coverage without moving up a level.

Cache Misses or Stale Content: Cache misses occur when requests fall outside of the cached regions. However, content staleness can only be determined based on heuristics, such as the number of times the content should have been refreshed since its last update. Currently, we assume content to be up-to-date. Some approaches to deal with missing or stale content could be to present the user with options to specify a new region for download or to update the stale content, while estimating the time and the bandwidth required based on download histories. Another approach would be to make a live request. It is an open question how to present these options in an application agnostic and understandable way to end-users. At this time, we rely on a live request for cache misses to ensure functionality.

5. EVALUATION

We implemented Caché as an Android service that runs in the background. Developers register their application on start-up with the service. Each registered application shows up as one of the privacy-enhanced applications under the Caché GUI. Through the GUI, the user can specify the content download region, set the content update rate and content priority. To reduce energy and bandwidth overhead, we have limited the service to only download content when the device is plugged in and a WiFi connection is available.

We evaluated Caché using two approaches. We first investigated tradeoffs in both download times and cache hit rates with respect to how much content was cached using two mobility datasets. Intuitively, download times should increase as more content is stored. Similarly, cache hit rates should also increase. Our goal was to understand how well Caché might work in practice. The other approach tells of our experience using Caché to enhance the privacy of three open source Android applications.

5.1 Evaluation Based on Mobility Datasets

We used real-world mobility traces from two different studies: Locaccino [36] and Place Naming [25], described in more detail below. Both sampled user location at approximately five minute intervals. We considered using some of the Crawdad[6] datasets, but did not find any that matched our needs.

Informally, our evaluation consisted of estimating the locations of a person's home and work, and then downloading all of the content within a certain radius. By adjusting the radius, we increase the amount of content that is prefetched. We then examined how many of the person's actual locations fall within these two radii to estimate how often Caché would provide cache hits.

Human Mobility Datasets: Locaccino is a location-sharing tool which offers users flexible control over who, when, and where one's location is shared with others. The Locaccino dataset has over 4000 people total. For our study, we selected the top twenty most active users of Locaccino, made up of graduate and undergraduate students, faculty, and a research software developer. Five of the selected users are female and the rest are males. The data was collected using a number of mobile clients, Android, Symbian, iPhone, and also laptops. The selected data consisted of approximately 460,000 location traces.

The Place Naming dataset consists of mobility traces collected in 2009: 33 users in the Spring, 26 users in the Summer, and 10 users in the Fall. The project collects data on how people label locations they visit. The participants consist of staff, undergraduates and graduate student, with ages ranging from 18 to 46. Over 40% of all participants are female. All of the data for the Place Naming dataset was collected using Symbian phones. The location data records GPS coordinates if satellite signal is visible, otherwise, a list of AP ids are recorded and translated to geo-coordinates using Skyhook at a later time.

Building the Cache: Our first method of evaluation focused on building a cache from scratch. We evaluate how the radius of the cached area affects the cache size and download time. For each user in our Locaccino and Place Naming datasets, we selected two significant locations. The significant locations are selected based on the frequency that they are visited and are required to be at least one mile apart. In the majority of cases, these two locations refer to the user's home and work places. However, we do not attempt to infer the relation of the significant locations to the participants. We downloaded content for the significant locations based on 5, 10, and 15 mile radii. The numbers are somewhat arbitrary, but reflect the fact that 15 mile radius covers most of the Pittsburgh area. The goal of this approach is to verify the amount of storage and bandwidth needed if the users were to build a fresh cache anchored at two significant locations which they frequently visit. The content downloaded is restaurant points of interest from MSN search and Yelp!

Table 3 shows the the size and estimated content download time. As the radius for the amount of area covered increases, user privacy is enhanced. The table presents the tradeoff between further enhancing user privacy with respect to cache size and the time required to build the cache. In the case of POIs downloaded from MSN and Yelp!, the amount of disk space and the time to download are not very large, as expected. Based on the results obtained, it is clear that the user can optimize for more privacy by downloading more content. We expect similar results with respect to other text based location data such as events, store locations, gas stations, and bus and movie schedules.

[6]http://crawdad.cs.dartmouth.edu/

Dataset	Radius (mi)	Size (kB)	Download Time (s)	Spatial Locality %	Spatial and Temporal Locality %
Locaccino	5	408	17	86.67	96.30
	10	620	26	86.75	96.66
	15	810	34	86.79	97.47
Place Naming	5	330	14	78.61	79.35
	10	384	16	83.75	84.75
	15	510	21	86.03	87.33

Table 3: This table shows the cache building statistics and hit rate. The content downloaded is restaurants from MSN and Yelp!. Download time is estimated based on a 200 kbps connection. Spatial Locality % refers to the approach where content is downloaded for two significant locations obtained through the study of each individual user's mobility data. Spatial and Temporal Locality % refers to the approach where, in addition to the user's hometown significant locations, two new significant locations are added to the user's download regions the day after they are visited.

Content Coverage: For this part of the analysis, we evaluate how the radius of the cached area affects the privacy of the user. To simplify analysis, we assume the user has fresh data. We used the Locaccino and Place Naming datasets to estimate the percentage of cache hits for the location-enhanced content. We evaluated cache hits by looking at the mobility trace of each individual user with respect to 5, 10, and 15 mile radii around the two significant locations. For each recorded GPS coordinate, we considered the entry a cache hit if it fell in either of the circles centered at the significant locations as defined previously.

Keeping in mind the above definitions for cache hits, we computed the cache hit rate for both datasets. As the hit rates are computed based on the user's two significant locations, the cache hit rates for the aforementioned approach are labeled *Spatial Locality* %. Table 3 presents the cache hit rates. It is noteworthy that by simply caching content at a 5 mile radius at two significant locations, more than 75% of the recorded GPS coordinates lead to cache hits. By caching at a 10 mile radius nearly 85% of the recorded GPS coordinates lead to cache hits. Although caching with very large radii increases user privacy, the increase in cache hits is marginal.

The approach based on caching content at two significant locations mainly caters to the scenario that the user is always unaware of plans to travel to a distant location. The cache misses correspond to users going out of the cached region bounds. However, we see that hit rate improvement, from 5 to 15 mile cache radius, although not negligible, is not very significant.

We explore another approach for when a user travels to a distant location, defined to be more than a 100 miles from home locations. Specifically, we cache content for the traveled location for future occurrences past the first day. This approach was based on the observation that if a user travels to a new location, for instance, from Pittsburgh to New York City, and stays for several days, then she can cache content once she arrives at the location for future days. As a result, on the first day when the user arrives, all recorded locations result in misses. However, on the second day at the same location, it is assumed that Caché downloads content for the new destination. As a result, all locations in the traveled destination that fall in the cached region will lead to cache hits past the first day. We refer to this approach as *Spatial and Temporal Locality*. By knowing that a destination has been visited and that it is likely that the user will be in the same location in the near future, we add a temporal locality concept to our location-enhanced content cache.

The results for the above are shown in Tables 3 under the Spatial and Temporal Locality % column. When the radius is 5 miles, the Locaccino dataset offers a substantially larger hit rate than the Place Naming dataset. We believe this is the result of also using laptops for recording locations using Locaccino. We find it likely that the bulkiness of laptops caused user location traces to be mostly captured in areas where the user can easily station herself. Thus, the user will be closer to known locations where she is more likely to comfortably use a laptop.

5.2 Porting Applications to Caché

In this section, we present our experience using the Caché Android service to improve location-privacy in three open source applications: mixare, Panoramio, and Restaurant Request. While all of the applications were written as standalone applications, we found the process of transforming the applications to run using Caché to require only minor modifications to the original source code. Mixare is an augmented reality engine browser, which presents Wikipedia, Twitter, and Buzz entries that are near the user's current location. Panoramio shows nearby pictures that have been uploaded by other Panoramio users. Finally, Restaurant Request is an open source application that we wrote that presents nearby restaurants overlaid on a map. Restaurant Request uses Yelp! as its content provider. The code for Restaurant Request is available on the Caché website[7].

For mixare, we focused on Wikipedia entries which fit the LTTL content description, i.e., the content is still useful even if cached once a day. The Twitter and Google Buzz entries fit the STTL description, where the value of cached entries diminishes rapidly. We selected a grid with overlay for mixare, because mixare uses a large radius to grab content, and content relevance would be unknown when the user is at the edge of the circular boundary. For Panoramio, we selected a grid with both the overlay and the hierarchy options, since content requests are based on the location and zoom level of the map as presented to the user. When both options are selected, the final content presented to the user is more accurate based on the user's map selection. Finally, for the Restaurant Request, we cached the nearby restaurant entries that were returned by Yelp!, using the default grid without overlay or hierarchy options. All of the data

[7] `www.ece.cmu.edu/~samini/projects/cache`

	SLOC		
Application	Original	Added	Removed
mixare	4692	18 (0.4%)	4 (0.1%)
Panoramio	1268	18 (1.4%)	7 (0.55%)
Restaurant Request	411	12 (2.9%)	8 (1.9%)

Table 4: Number of Source Lines of Code that had to be added to and removed from each of three sample Android applications to run using the Caché service. Any source code provided as a template is not taken into account.

was cached for a 5 mile radius centered at the CMU campus and updated on a daily basis.

We have made the process of connecting to the Caché service simple by writing an interface package that can be added to any Android project. We also have a template that a developer can copy into the application source code to connect using the interface code. The template code is 30 lines and is not counted towards actual source code change. Table 4 presents the lines of code that we had to add to and remove from each application. All three applications required less than 5% total code change, with 16 lines of code added, and 6.4 lines of code removed on average. Most changes involve routing the request to Caché rather than the Android HTTP stack.

Based on our experience, applications can be transformed into Caché-enabled applications easily without major code change and without in-depth knowledge of the internal workings of Caché.

6. DISCUSSION

In this section, we discuss limitations, developer burden, adoptability, and further work.

Limitations: Caché targets certain LTTL content as discussed previously. Therefore, mobile services that require rapidly changing content or user interaction with a server are outside of Caché's model. For instance, Caché would not work for location check-in applications such as Foursquare [26], which requires the user to check-in with a server at a particular location.

Caché expects the user to have requested content for regions she is likely to visit. If this is not the case, the user has to rely on other privacy-enhancing approaches or to make a live request. It is also important to consider what the real cost of a cache miss could be. For instance, it may be okay for a user if a restaurant has relocated to another location and the user is forced to choose another option. However, if a bus schedule is stale and the user is in a rush to get to a location, the cost of a cache miss could be more expensive. We conjecture that if the user's downloaded content is small enough to be downloaded completely over night, for instance on the order of hundreds of MBs, then the content could be downloaded every night. As a result, the chance that an expensive cache miss would occur would be considerably reduced.

Developer Burden: The application developer is responsible for defining the cell size in the content download grid, content update rate, and content priority. If the developer does not have a good understanding of the content and application domain, then the grid created by Caché could be inefficient in grabbing the content that users expect. When in doubt, it is best for the developer to select a small grid size with both the overlay and hierarchy options included. This way, Caché can present the correct content based on the size of the region for which content is requested. For cases where the LBS requires a query string, e.g., *restaurants*, *pizza*, *libraries*, the burden of correctly guessing the user's query falls on the developer. This may lead to an incomplete sweep of data that the user expects.

Defining content download parameters requires knowledge about content density. Although Caché requires the developer to have more of an understanding about the application content, we conjecture that developers have such insights already. For instance, in our examination of the mixare soure code, we noticed that 50 rows of content were requested for a 20 km radius. The examination of the Panoramio source code revealed that the application presents 20 photo entries to the user at a time. Developers make decisions regarding how much content to request and how to present content to users already. However, Caché would require developers to understand such decisions with an additional level of abstraction.

Adoptability: Caché does not require additional infrastructure to enhance user privacy. Further, caching and pre-fetching are simple and familiar concepts to developers. However, instead of relying on pre-fetching and caching to improve performance or to enable content use on disconnected or weakly connected devices, Caché uses them to enhance user privacy. Of course, this does not diminish the benefits of pre-fetching and caching. Instead, it makes Caché appealing for developers as it boosts application performance while reducing the burden to make the application privacy friendly. In other words, unlike most approaches to location privacy, Caché provides developers with a strong incentive to increase privacy because pre-fetching and caching improve performance and energy efficiency, and enable disconnected operation. In a practical sense, our approach has a higher chance of being adopted versus complex privacy preserving solutions that require additional infrastructure or developers with expert knowledge of preserving user privacy.

Further Optimizations: Caché can be optimized to learn the content update rate and prioritization. Ideally, there would be a protocol that lets end-users download only updates from content providers in bulk. However, there is currently no such protocol. An alternative is to update content based on an exponential back-off approach. Using such an approach, Caché could use the update rate requested by the developer as a guide rather than canonical. The same is true for content prioritization. If the user uses the data for an application more often than others, the content prioritization for that application can be increased. Note that the information leak due to prioritization is minimal, as it does not change the update rate of the content, only the order in which it is downloaded.

Caché can further be optimized by sharing content between applications that use the same content provider. The caveat here is that the shared content should be using the same service with the same API calls and similar cell sizes in the grid. The system then has to be extended to recognize when the aforementioned conditions hold, and to service the requests of one application with another application's content.

Another aspect of Caché which may be optimized is how content is downloaded for a large region. One could imagine that instead of downloading content sequentially or randomly, content for hot spots or popular locations could be downloaded first. Another option would be to download such that an overview is available early on, so that the user has some useful data for various regions before the entire grid is downloaded.

7. RELATED WORK

In this section, we present related work to the Caché approach. We start with content caching and proceed to discuss location privacy.

Content Caching: Caching content has been a well-explored topic for mobile computing, primarily focused on performance or disconnected or weakly connected devices. For example, the Bayou architecture is designed from the ground up to support mobile computing applications [8], while the Coda file system is designed to provide support for weakly connected operation [20, 37]. Our work differs from this previous work in that we apply caching for location privacy, examining the tradeoffs of such an approach in today's Internet architecture.

The content used by Caché is obtained through existing web sites and content providers on the Internet. As with other caching solutions, the content may become stale depending on its update frequency. Cache staleness is a well-known problem and has been explored in many domains, for example in web search and indexing [6, 31, 33, 43]. Our work does not directly address this problem. Instead, we provide an analysis showing that there are many types of location-enhanced content that can be effectively cached for reasonable periods of time while still remaining accurate.

Location Privacy: Many past projects have explored balancing the tradeoffs involved in providing useful functionality while offering privacy protection to end-users. Recent work by Brush looks at if and how users understand the effectiveness of various privacy preserving techniques and how much users value their location data in monetary terms [5].

Iachello and Hong offer a summary of current and future research trends in privacy in their survey [17]. Duckham and Kulik [10] sketch out four major themes for location privacy, namely regulation, privacy policies, anonymity, and obfuscation. Krumm [22] offers a survey of computational approaches to location privacy that examines inference techniques as well as countermeasures. Issues surrounding regulation and privacy policies are beyond the scope of this paper. Instead, we will focus on the other two areas.

Anonymity: A common theme in much of the work on anonymity has been relying on a trusted third party that acts as a proxy between the client and the location-based service. One metric that has been developed is k-anonymity [11, 40]. There are many examples of work in location privacy using k-anonymity. Perhaps the most relevant here are spatial and temporal cloaking [12], Mix Zones [13, 4], New Casper [29], and (to some extent) CacheCloak [28]. Recent work by Shokri questions the effectiveness of k-anonymity for preserving location privacy with a common misunderstanding of confusing query anonymity and location privacy [39]. Our work here differs in that Caché does not focus on anonymity. Instead, Caché offers a different model for accessing location-enhanced content, one that relies on pre-fetching and disconnected operation.

Privad [14] is an online advertising system that anonymizes ad requests and ad view/click reports using untrusted ad dealers. Caché is similar to Privad with respect to pushing content storage and processing to the client. Anonymizing networks may also be used to preserve user privacy. Tor [9] is one practical implementation of a low-latency anonymizing network. However, the performance issues associated with Tor, specially in mobile applications, compromise application usability. Further, Tor does not protect against services that require authorization. With respect to anonymity approaches, Caché does not require any third party, nor does it require a critical mass of users to be effective.

Anonymity can also be established with encryption techniques. Private Information Retrieval (PIR) [7, 23, 32] allows clients to make queries to a server in a way that the server cannot distinguish which memory address was read. The approach is interesting, however, the support would need to be implemented both on client and server side, and the use of cryptography would introduce additional overhead to the system.

Obfuscation: Many techniques have been developed, including adding noise, quantizing locations (essentially putting locations into buckets or aligned onto a grid), and adding false locations. SybilQuery is a client-side tool that creates many different queries to the server to obfuscate the user's actual path [38]. Caché relies on obfuscation in that it only shares with content providers that the user is in a geographic region, e.g., a neighborhood or a city. However, Caché uses obfuscation in a different way than past work, in that after retrieving content for a region, it processes and filters content locally on the user's mobile device. Thus, content providers only know that the user is in a large region rather than obfuscated trails or locations.

Privacy-Enhancing Systems: Finally, we discuss two systems that are perhaps the closest to Caché in terms of goals. Confab is a framework with the purpose of providing support for building ubiquitous computing application with privacy enhancing mechanisms [16]. Caché is a logical extension of this past work as it vastly expands the kind of data types available for application development. Caché also provides deeper analysis of the tradeoffs involved. Another approach for hiding the user's location by surrounding it with other users' paths is CacheCloak [28]. CacheCloak caches all previous requests and services queries from the cache first. If data is not available, it makes a live query, disguising the user's current location by requesting data along the predicted path, extended until the path intersects with other paths. CacheCloak shares similar design ideas to our approach. However, predicting mobility is not always necessary. We show that pre-fetching is feasible for a substantial amount of data without hindering system usability.

8. SUMMARY

We presented the feasibility analysis, design, implementation, and evaluation of Caché, a system that lets people access location-enhanced content while offering location privacy through the use of pre-fetching.

In our feasibility analysis, we examined the privacy model, as well as familiar systems issues of caching, data freshness, and data consistency. We provided a taxonomy of location-enhanced content, organizing content types as short-time-to-live (STTL) and long-time-to-live (LTTL), and showed many examples of content that could realistically be LTTL, and thus, could potentially be pre-fetched. We estimated how often various web sites changed their content, showing that several kinds of data types could be cached and still useful for at least one day (if not longer). We also provided estimates of how much storage and bandwidth would be required for Caché, showing that for map tiles and basic text-based location-enhanced content, content could be easily stored and downloaded using existing devices and network connections.

Finally, we provided two evaluations of Caché. One evaluation was based on the examination of tradeoffs with respect to two human mobility datasets from separate user studies. First, we showed the tradeoff between the size of the geographic region for which to download content and the size of the cached content and download time. Second, we examined how often people's locations would place them in an area where they had cached content, adjusting the size of the cached geographic region and seeing how things changed. In both cases, we presented results showing that one does not need to cache a great deal of content for the system to be effective, strongly suggesting that caching location-enhanced content is a feasible strategy to improve user privacy. Our second evaluation was based on our experience using Caché to enhance user privacy in three Android applications. The average change in the source lines of code was 16 lines added, and 6.4 lines removed. No in-depth knowledge of Caché was necessary to make the privacy enhancements.

9. ACKNOWLEDGMENTS

We thank Mahadev Satyanarayanan, Dan Siewiorek, and Asim Smailagic for their generous constructive feedback in supporting this work. We also thank our shepherd, Landon Cox, for his insightful feedback. This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273 from the Army Research Office. This research was also supported by NSF CNS-0627513. Additional support has been provided by Microsoft through the Carnegie Mellon Center for Computational Thinking, FCT through the CMU/Portugal Information and Communication Technologies Institute, and through grants from FranceTelecom, Nokia, and the Alfred P. Sloan Foundation. Janne Lindqvist is supported by Academy of Finland, and the foundations Emil Aaltosen Säätiö and Tekniikan Edistämissäätiö.

10. REFERENCES

[1] S. Amini, J. Lindqvist, J. I. Hong, J. Lin, E. Toch, and N. Sadeh. Caché: Caching Location-Enhanced Content to Improve User Privacy [Extended]. Technical Report CMU-CyLab-10-019, CMU Cylab, 2010.

[2] S. Amini, J. Lindqvist, M. Mou, R. Raheja, J. I. Hong, J. Lin, E. Toch, and N. Sadeh. MobiCom 2010 Poster: Caché: Caching Location-Enhanced Content to Improve User Privacy. *ACM MC2R*, 2010.

[3] L. Barkhuus and A. Dey. Location-Based Services for Mobile Telephony: a study of user's privacy concerns. In *Proc. of Interact*, July 2003.

[4] A. R. Beresford and F. Stajano. Location Privacy in Pervasive Computing. *IEEE Pervasive Computing*, 2(1), 2003.

[5] A. B. Brush, J. Krumm, and J. Scott. Exploring end user preferences for location obfuscation, location-based services, and the value of location. In *Proc. of UBICOMP*, 2010.

[6] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. *SIGMOD Rec.*, 29(2), 2000.

[7] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *Proc. of FOCS*, 1995.

[8] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing Among Mobile Users. In *Proc. of WMCSA*, Dec. 1994.

[9] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. of the 13 th Usenix Security Symposium*, 2004.

[10] M. Duckham and L. Kulik. *Location privacy and location-aware computing*, chapter 3. CRC Press, 2006.

[11] B. Gedik and L. Liu. Protecting location privacy with personalized k-anonymity: Architecture and algorithms. *IEEE TMC*, 2007.

[12] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proc. of MobiSys*, 2003.

[13] M. Gruteser and B. Hoh. On the Anonymity of Periodic Location Samples. In *Proc. of SPC*, 2005.

[14] S. Guha, B. Cheng, and P. Francis. Privad: Practical Privacy in Online Advertising. In *Proc. of NSDI*, 2011.

[15] J. Hong, G. Borriello, J. Landay, D. Mcdonald, B. Schilit, and D. Tygar. Privacy and Security in the Location-enhanced World Wide Web. In *Proc. of UBICOMP*, 2003.

[16] J. I. Hong and J. A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proc. of MobiSys*, 2004.

[17] G. Iachello and J. Hong. End-user privacy in human-computer interaction. *Found. Trends Hum.-Comput. Interact.*, 1:1–137, January 2007.

[18] Y. Jung, P. Persson, and J. Blom. Dede: design and evaluation of a context-enhanced mobile messaging system. In *Proc. of CHI*, 2005.

[19] W. Karim. Privacy Implications of Personal Locators: Why You Should Think Twice before Voluntarily Availing Yourself to GPS Monitoring. *Washington University Journal of Law and Policy*, 14(485), 2004.

[20] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda File System. *ACM Trans. Comput. Syst.*, 10(1), 1992.

[21] J. Krumm. Inference attacks on location tracks. In *Proc. of PERVASIVE*, 2007.

[22] J. Krumm. A survey of computational location privacy. *Personal Ubiquitous Comput.*, 2008.

[23] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proc. of FOCS*, 1997.

[24] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, T. S. James Scott, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place lab: Device positioning using radio beacons in the wild. In *Proc. of Pervasive*, 2005.

[25] J. Lin, G. Xiang, J. I. Hong, and N. Sadeh. Modeling people's place naming preferences in location sharing. In *Proc. of UBICOMP*, 2010.

[26] J. Lindqvist, J. Cranshaw, J. Wiese, J. Hong, and J. Zimmerman. I'm the Mayor of My House: Examining Why People Use foursquare - a Social-Driven Location Sharing Application. In *Proc. of CHI*, 2011.

[27] MapPoint. Microsoft mappoint 2009, north america. *http://www.microsoft.com/mappoint*, 2009.

[28] J. Meyerowitz and R. Roy Choudhury. Hiding stars with fireworks: location privacy through camouflage. In *Proc. of MobiCom*, 2009.

[29] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new Casper: query processing for location services without compromising privacy. In *Proc. of VLDB*, 2006.

[30] D. Narayanan and M. Satyanarayanan. Predictive Resource Management for Wearable Computing. In *Proc. of MobiSys*, 2003.

[31] C. Olston and S. Pandey. Recrawl scheduling based on information longevity. In *Proc. of WWW*, 2008.

[32] F. Olumofin, P. K. Tysowski, I. Goldberg, and U. Hengartner. Achieving efficient query privacy for location based services. In *Proc. of PETS*, 2010.

[33] S. Pandey and C. Olston. Crawl ordering by search impact. In *Proc. of WSDM*, 2008.

[34] C. Pettey. Gartner Says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011. *http://www.gartner.com/it/page.jsp?id=1529214*, Jan 2011.

[35] Pols. Privacy Observant Location System. *http://pols.sourceforge.net*, 2008.

[36] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao. Understanding and capturing people's privacy policies in a mobile social networking application. *Personal Ubiquitous Comput.*, 13(6), 2009.

[37] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu. Experience with disconnected operation in a mobile computing environment. In *Proc. of MLCS*, 1993.

[38] P. Shankar, V. Ganapathy, and L. Iftode. Privately querying location-based services with SybilQuery. In *Proc. of Ubicomp*, 2009.

[39] R. Shokri, C. Troncoso, C. Diaz, J. Freudiger, and J.-P. Hubaux. Unraveling an old cloak: k-anonymity for location privacy. In *Proc. of WPES*, 2010.

[40] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5), 2002.

[41] D. Turner. Broadband Reality Check: The FCC ignores America's Digital Divide. *Consumer Union*, 2005.

[42] S. Wang, J. Min, and B. Yi. Location Based Services for Mobiles: Technologies and Standards. In *Proc. of ICC*, 2008.

[43] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proc. of WWW*, 2002.